TOWARDS ENERGY-EFFICIENT QOS-AWARE ONLINE STREAM DATA PROCESSING FOR INTERNET OF THINGS

THE UNIVERSITY OF SYDNEY

A thesis submitted in fulfilment of the requirements for the degree of Master of Philosophy in the School of Information Technologies at The University of Sydney

Yajie Zhang

July 2017

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Yajie Zhang
March 2017

Abstract

Online data stream processing in Internet of Things (IoT) systems is an emerging paradigm that allows users to use resource-constrained IoT devices with the back-end of resourceful machines to process the data collected from the physical world in a real-time manner. The huge amount of generated sensor data can produce value-added information with different purposes for several applications. Techniques to promote knowledge discovery from the raw data allow fully exploiting the potential usage of wide spread sensors in the IoT. In this context, using the energy of the resource-constrained IoT devices in an efficient way is a major concern. However, the application of QoS requirements should not be ignored to achieve the purpose of energy saving at any cost. In this thesis, we propose a framework that combines online stream data processing with adaptive system control to address both needs. The online algorithms are based on statistical methods to meet the needs of stream data processing. The result of the algorithms are then used to dynamically control the system behaviour to meet the needs of energy-saving. Simulation results show the effectiveness of our proposed framework.

To My Beloved Parents
For Their Endless Support
To Myself In The Future

Acknowledgements

By the time I complete this thesis, what I have learned is more than just the word of research.

I would like to thanks everyone that have helped me with the work, with the thesis and with all the problem I met during the master years. But more importantly, I would like to thanks some special people that mean a lot to me expect for the thesis.

To my supervisor, Albert Zomaya from the Centre for Distributed and High Performance Computing of the School of Information and Technology at the University of Sydney. Thanks for the support all the time. You are a big boss, and you like the word boss, but you are so kind to us. Have you as the supervisor feels like no one will ever close the door for me, kind of feelings of being protected. The time we spend together were short, realizing that I haven't given myself a chance to talk to you other than a student of you. Wish the chance in the future for a conversation in-depth.

To my co-supervisor Wei Li from the Centre for Distributed and High Performance Computing of the School of Information and Technology at the University of Sydney. Memory back to the first and second time I met with you, the first thought was surprise, the second thought was further surprise. We were from the same university, which feels like a magical connection. And you are so wise and patient. You understand my every wired ideas or concerns, and you have your own way to response. You support me very much in the work, how to make myself understood, how to solve the problem. You cleared out my mind when I walked myself into some dead ends. Personally, you are clear about everything in mind, but you are just so kind that you don't point something out. If there is no you, everything will be a lot harder. Also, because of you, I always regard things easy, and afraid of nothing, cause you will be there, fixing everything, even a tiny reference number. You are supervisor, as well as friend, as well as family in this place. But sometimes just because of this, I am afraid of letting you down, though I know still a lot. In all, it is my luck to meet you! Thanks you.

Then, many thanks to Flavia Delicato and Paulo Pires from Federal University of Rio de Janeiro. It is very lucky to have the master years when you are visiting here. You two taught me a lot on the research and on the work. You remind me a lot of things I have never noticed, in the research, in the paper, which are far away from just useful in the papers. I learned a lot on how to be responsible to my own work and to the people who will read my work. And Flavia, you impressed me with your rigor to the work. You are so careful and serious with everything that you help me, which means a lot to me. But besides research, you and Paulo are kind, and love the life.

What's more, thanks for the help of Mr. Hanjun Ma. You have helped me a lot, when I was lazy, when I was not good at something. You have covered many trivial but important troubles that I threw to you. And thanks for reading my work so carefully and provide me suggestions so carefully.

Finally, thanks to my parents for everything. Word is never enough to express what you, my mother and father, mean to me and provide me. I miss you.

List of Publications

Yajie Zhang, Wei Li, Flavia C. Delicato, Paulo F. Pires, Albert Y. Zomaya, Claudio M. De Farias, Luci Pirmez. ”Towards Energy-Efficient QoS-Aware Online Stream Data Processing for Internet of Things.” European Conference on Parallel Processing, 2017 (under review).

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Idea and IoT

Internet of Things (IoT) [1] aims to interconnect physical objects of all kinds, from people and body parts, to huge civil structures, e.g. buildings, bridges and oil rigs. Wireless sensor and actuator networks (WSAN) [2] lay the foundation of the Things layer in IoT systems. These objects will be embedded with small devices, which are typically battery-powered and equipped with electronics, software, sensors, actuators, and network connectivity that enable them to collect and exchange environmental data and to perform simple actions upon the physical world. Such instrumented objects or things are called smart objects, and their integration with the Internet will enable an ecosystem of smart applications and services that will improve and simplify the life of the citizens and contribute to sustainable growth. Hence these devices are responsible for collecting the environmental data, performing simple processing on such data and then communicating with higher level systems, e.g. the edge nodes or the cloud platforms, which can perform more sophisticated computation and data analytics if required. Typically, sensors collect the physical environment data over a specified time period to identify phenomena or events of interest to the application.

The nature of the physical world is intrinsically changing, and these changes are reflected in the data collected and the models built from these data. Due to this feature, the collected data could be burst, noisy, and unpredictable and thus, it is best modeled as transient data streams in contrast with the traditional persistent relations of typical information systems. The distinctive feature of the data stream model relies not on the structure of each individual data items, which is the same as in traditional models, but in the stream nature of the produced data: unbounded in size, continuous arrival in multiple, rapid, time-varying and possibly with unpredictable behavior [3].

Data stream processing systems typically require using on-line solutions (one that computes its output using only a single in-order scan of its input to avoid unbounded buffering while still guarantees the result can be calculated in a numerically stable fashion.) since the valuable information can be acquired on the fly without having the complete view or even a priori knowledge of the data. The ability to identify changes in the input data is essential to the success of a stream processing system, so that the knowledge discovery can be promoted from the raw data in early stage. Changes may denote an event of interest that consists in general data patterns, trends in the data distribution, and provides insightful information that can then be used in further stages of application-specific decision making. At the same time, the early findings on the data behaviour can be leveraged for the purpose of saving the precious resources of IoT devices (throughout this paper, IoT devices and sensor nodes are used interchangeably), increasing the overall cost effectiveness of the system.

1.2 Edge Computing

Data is increasingly produced at the edge of the network, therefore, it would be more efficient to process the data at the edge of the network. A number of approaches, such as micro Datacenter, Cloudlet, and fog computing have been introduced to the community since Cloud computing is not always efficient for data processing when the data is produced at the edge of the network.

A new approach called edge computing has been introduced to the community recently. It refers to the enabling technologies that allow computation to be performed at the edge of the network in a stream fashion, including both the downstream data process on behalf of cloud services, and upstream data process on behalf of sensing data.

According to [4], the reasons for introducing edge computing can be illustrated by following points:

1) The computing power of the cloud cannot cover the increasing needs of the real world applications, and in some case the data needs to be processed at the edge for

shorter response time, more efficient processing and smaller network pressure. Thus, it requires the edge computing to push data from cloud services.

2) The huge amount of data cannot always upload to the cloud in a timely manner, so it needs to be processed at the edge of the network, which is to pull data from Internet of Things.

3) Data producer can also be data consumer. A lot of data come from the producer need to be processed at the same time. The change from data consumer to data producer/consumer requires more function placement at the edge. Processing data at the edge of network can also help to protect user privacy.

Edge computing can thus better handle the real-time data processing requirements, especially when dealing with large, real-time data streams such as data streams generated by IoT devices. As a result, edge computing is increasingly used in the IoT systems.

1.3 Data Stream Processing

A data stream is a real-time, continuous sequence of items with the definition stated above [5]. In regard to the features of the stream data, there are some unique requirements in processing them [6]: 1) the procedure should have no requirement to store the whole data in the stream to conduct any operation or sequence of operations. 2) there are built-in mechanisms to action against stream imperfections, including delayed, missing or out-of-order data. These data are commonly occurred in real world data streams. The procedure should not be allowed to have unnecessary wait in delayed or missing data, which will block or delay the computation. However, when missing data causes any information loss, the procedure should be able to be aware of it. Besides, an updating mechanism should be involved to deal with out-of-date data. 3) A data stream is never going to be a whole data set, but the data items are still used to represent knowledge. Thus, the processing engine should be able to generate predictable outcomes which are used to carry out required operations for applications. 4) Based on the previous requirements, each outcome can be regarded as temporary and old result in the view of the long term run. Thus, the procedure should be able to intelligently store, access and manage these state outcomes. In this case, the procedure will be able to monitor the situations or compare and combine the information form the previous outcomes with the information from current stream. 5) In related to the resource and efficiency requirements of edge computing, data stream processing requires an automatically distribution across multiple processors to partition and scale the applications. 6) There should be an optimized balance between quality and the response time of the results based on applications. Usually, the execution should respond instantly and the correction action is taken during the long-term run.

Generally, regarding to the actions and communications between billions of devices that produce and exchange data related to real-world objects of Internet of Things, extracting higher level information from the raw sensory data captured by the devices and representing this data as interpretable information, for both the machine and human, has several interesting applications. Deriving raw data into higher level information representations demands mechanisms to find, extract, and characterize meaningful abstractions from the raw data. These meaningful abstractions then have to be presented in a human and/or machine-understandable representation. Our proposal targets on the low level processing of raw data.

1.3.1 Statistical Methods

Statistical methods allow us to collect, describe, analyze, and draw conclusions from data. They can be used in a purely exploratory context to describe properties of the data, but also as estimators for model parameters or in the context of hypothesis testing. For example, for the data distributions that are approximately symmetric, the mean value is used to indicate the measures of center, which describes where the data distribution is located along the number line. However, it is also an estimator for the expected value of a probability distribution from which the data are sampled.

When data distributions are no longer symmetric, e.g. they are noticeably skewed or have outliers, the mean and the variance are not the best options for describing the measures of center and spread. Instead, the higher order moments, e.g. skewness and kurtosis, are the better ones to use to mathematically describe such distributions. Skewness provides the information about the asymmetry of the data distribution and kurtosis gives an idea about the degree of peakedness of the distribution. More precisely, skewness describes the problem of data asymmetry that whether or not a certain variable is symmetrically distributed around its mean value. This indicates that an asymmetrical frequency curve may arise in two cases: 1) the case of homogeneous material when the tendency to deviation on one side of the mean is unequal to the tendency to deviation on the other side; and 2) the case of heterogeneous material consisting of a mixture of two or more homogeneous materials.

The outcome of kurtosis addresses the structure of the distribution and informs how much the data is peaked or flat relative to a normally distributed dataset. The reference standard is zero by using the normal distribution whose excess kurtosis equal to kurtosis minus three as reference. And further, we can define certain dataset which have a highly dispersed data as platykurtic, or one that have the data with higher frequency of values concentrated near the mean leptokurtic, or one with the data concentrated around its mean like a normal distribution mesokurtic. Thus for a platykurtic dataset, the distribution results in a graphical dispersion of lower peak (lower kurtosis) than the curvature found in a normal distribution. And for leptokurtic dataset, the observation shows a sharp peak that is more extreme than a normal distribution (excess kurtosis exceeding zero).

Therefore, statistical methods are widely used for computation of similarities or patterns in WSAN. Most works usually focus on the mechanism of data analysis but lack of concerns about the computation capability, efficiency and resource cost. In [7] the authors introduce an information fusion algorithm based on the statistical concepts of Kurtosis and Skewness. They proposed techniques to promote knowledge discovery from the huge amount of sensing data are required to fully exploit the potential usage of WSAN. Employment of such techniques are useful to reveal trends in the sampled data, uncover new patterns of monitored variables, make predictions, thus improving decision making process, reducing decisions response times, and enabling more intelligent and immediate situation awareness. However, it is a centralized two pass process (explained in the following) for calculating the statistical results which does not perform well with large data sets in resource constrained IoT devices.

The traditional statistical methods of calculating the statistical moments are known as the two-pass (a.k.a. off-line) algorithm in the field of computing. They are called two-pass algorithm since the mean value must first be computed and it will be subsequently used in the computation of the higher order moments, which implies the entire dataset must be retained for every computation. Due to the nature of two-pass algorithms require large storage space and computation resources for big datasets, they are not suitable to be used in resource constrained devices.

One-pass (a.k.a. on-line) algorithm can be employed to tackle the aforementioned issues with conventional two-pass algorithms. In general, a one-pass algorithm is one that computes its output using only a single in-order scan of its input to avoid unbounded buffering while still guarantees the result can be calculated in a numerically stable fashion. A one-pass algorithm generally requires at most O(n) time and less than O(n) storage (typically, O(1)). Thus, it is more suitable to be used on those resource constrained devices for analyzing large-scale datasets on the fly.

1.3.2 Performance Guarantee

To guarantee the accuracy of the outcomes, monitoring mechanisms are widely used in data stream processing systems. Distribution or value changes may occur in data stream after a certain period of time, or the distribution of data may be incomplete before a certain period especially for which generated from dynamic and complex environments or applications, yielding different or incorrect processing outcomes. Hence, the system should be aware of changes in the data or the differences in the outcomes. Therefore, change detection method is introduced into the systems and is widely used in monitoring data processing outcomes especially for higher level processing such as knowledge discovery. Generally, change detector methods continuously monitor the outcomes by comparing the results, deleting useless results, merging useful information and carrying out corrective actions [7].

Early change detectors apply statistical probability methods to compare the intermediate or final outcomes directly and see if there is change in the results. These advanced probability methods are highly accurate. However, they are lack of adaption ability in modern data processing. Therefore, resent change detectors implement feedback detection method on the outcomes. In detail, the outcomes are regarded as prediction outcomes as described above. Thus, new data provide the reference and decide the accuracy of the predictions. And then, based on the real case reference, the procedure correction itself in further processing. In this case, the error rate of the outcomes is obtained. It is regarded as stable if the error rate keeps or decreases because of the correction actions. That is to say, if there is a significant increase in the error rate, it is possible that the situation is changing and the procedure should take actions in regard to the changes. Such change detectors, such as: ADwin [8], SeqDrift [9] are widely used and well evaluated.

1.4 Proposal and Contribution

Realizing the full potential of IoT systems requires the development of novel software technologies to enable the deployment and operation of wireless sensing and control systems. However, the needs of the effective processing of generated data streams and the efficient usage of the available resources of devices are normally addressed as two separated issues. In the context of IoT, considering both needs as a whole is the key to efficiently process IoT data streams.

To tackle this issue, we propose an online data stream processing approach tailored to the needs of IoT applications based on statistical methods. Statistics methods are adopted for their simplicity of implementation and low computational cost. Moreover, we use the results from our data stream processing approach as the feedback signal for adaptively controlling the resource usage on IoT devices within the system. By doing this, our framework can decrease the data transmission of sensor nodes according to the data stream behaviour, thus saving energy. Meanwhile, the adaptive adjustment of the system resource usage is triggered at the edge nodes whenever it is necessary to meet the accuracy constrain posed by applications, which ensures sufficient sensor nodes are used for data measurements.

In this thesis, we introduce a basic framework and an enhanced framework as the solutions for the issues stated above. For each framework, the purpose of self-adaptive QoS-aware energy efficient data stream processing is achieved in different way.

1.5 Thesis Structure

The rest of the thesis is organized as below. In **Chapter2**, We present the state of art of the data stream processing system in the context of IoT. And then in **Chapter3** and **Chapter4**, we introduce our basic solution and our enhanced solution. For each solution, the system model and architecture, algorithm design, and experiment are provided. In addition, We compare the performance of two solutions at the end of the **Chapter4**. Finally, in **Chapter5**, we draw a conclusion on the work presented in the thesis and discuss several possibles ways to extend our work.

Chapter 2

Literature Survey

Considering the WSAN data stream processing under IoT context, there are proposals regrading to different aspects. To tackle the issue we presented above, there are mainly two areas of works which are about: 1) resource utilized system in WSAN targeting on such as energy, storage, or life time limitations, under constrain of QoS requirements such as accuracy, and 2) efficient large data stream processing targeting on the heavy or over workload, the efficiency or the real-time response time requirements of the system when facing large amount of the input data. Compared with and inspired by these works, our thesis aims at filling the gaps these solutions have in tackling the issue.

2.1 Data Processing

Statistical methods are widely used for computation of similarities or patterns in WSN. Most works usually focus on the mechanism of data analysis but lack of concerns about the computation capability, efficiency and resource cost. In [10] the authors introduce an information fusion algorithm based on the statistical concepts of Kurtosis and Skewness. It learns the Kurtosis and Skewness value and based on the features for the shape of the kurtosis and skewness. In detail, the shape of the two values indicate a rough information on the situation happens in the real application. However, it is a centralized two pass process for calculating the statistical results which does not perform well with large data sets in resource constrained IoT devices.

However, the approaches above are both based on the data processing parallelization which are not a direct transmission control in regard to the high volume of data. While there are congestion control in regard to the high volume of data in TCP/IP area [[11]]. In the work [11], the author introduces an advanced congestion control function based on cubic function. Based on the transmission flow and load, the protocol adjusts the threshold for the transmission. And the implementation of cubic function aims at keeping the transmission allowance around the maximum as long as possible, in order to fully utilize the transmission resources. The same control idea can be applied to the wireless sensors networks to increase or decrease the transmission when possible.

2.1.1 Adaptive Data Processing

In respect to the large amount of data in data stream in WSAN networks, there are proposals dealing with the possible transmission or processing congestions. Most works consider the solutions in related to distribution and task scheduling methods. In [12] the authors provide an elastic auto-parallelization scheme that can handle stateful operators that works across multiple hosts. A control algorithm is applied periodically reevaluating the number of channels to be used based on local run time metrics it maintains. The control algorithm is designed to decide the increase or decrease the number of channels used or take no action at any decision point. When the number of channels to use changes, then a state migration protocol may need to be executed if the parallel region is stateful. However, it is a state based solution, which is not suitable for WSANs, especially when data transmitted from each sensor is not always at the same time. That is to say, it is hard to define the state.

The proposal in [13] describes a framework for real-time semantic annotation of streaming IoT data to support dynamic integration into the Web using the Advanced Message Queuing Protocol (AMPQ). The real-time annotation framework aims to semantic annotation of IoT stream data by taking into account dimensionality reduction and reliability. This enables the delivery of large volume of data that can influence the performance of the WSANs under IoT context.

2.2 Resource Constrained WSAN System

There is a large amount of papers proposed recent years about energy saving in WSANs.

Firstly, energy efficient system are widely acquired in different applications especially in industry applications. These applications include Health Care, Manufacturing and smart grids, Transportation systems, environment, and etc.

In such applications, the system requires the sensors operating autonomously for a long period of time, ranging from weeks to months. However, every application is constrained in terms of energy because of the limited battery resources of the sensors that limits the network lifetime. In fact, it is usually impossible to manually replenish the sensor motes because of their number, the maintenance cost or the inaccessibility of monitored regions. Besides, some applications such as health care can tolerate battery replacement. Although, we believe that the rapid depletion of the battery prevent their wide adoption. Thus in order to achieve energy efficiency, we present several existing developments. One way is the automation of monitoring and control systems which is an important purpose for many utility companies in manufacturing, water treatment, electrical power distribution, and oil and gas refining.

The Smart Grids is aiming to monitor the energy supply and consumption process based on the automated and intelligent power system managing mechanism. Potential application of sensor networks in smart grids are: sensing the relevant parameters that affecting power output (temperature, humidity, wind orientation, radiation, etc.); remote detection of faulty components; control of motors and underground cables; and home energy management [14].

Then regarding to the detail mechanisms on energy efficiency of these application, existing major methods include:

1) Radio optimization: The radio module is the main component that causes battery depletion of sensor nodes. To reduce energy dissipation of wireless communications, researchers have tried to optimize radio parameters such as coding and modulation schemes, power transmission and antenna direction. Modulation optimization is aiming at finding the optimal modulation parameters that result in the minimum energy consumption of the radio. For example, Costa and Ochiai [15] studied the energy efficiency of three modulation schemes. And derived from this, the modulation type and its optimal parameters that achieve minimum energy consumption for different distances between nodes. Transmission Power Control (TPC) has been investigated to enhance energy efficiency at the physical layer via adjusting the radio transmission power [16].

In CTCA (Cooperative Topology Control with Adaptation) [17], the authors intend to regularly adjust the transmission power of every node in order to take the uneven energy consumption profile of the sensors into consideration. That is to say, a node with higher remaining energy will increase its transmission power. In this case, the other nodes are enabled to decrease their transmission power, and thus saving energy. However, TPC strategy effect not only on the energy but also on the delays, link quality, interference and connectivity [18]. When transmission power decreases, the risk of interference also decreases. While on the contrary, delay is potentially increased because more hops will be needed to forward a packet. Finally, transmission power influences the network topology because the potential connectivity between sensors varies.

2) Data reduction: Another approach aims to reduce the amount of data being delivered to the sink node. Two methods can be adopted jointly: the limitation of unnecessary samples and sensing tasks because both data transmission and acquisition are costly in terms of energy.

Firstly, in data aggregation schemes, nodes along a path towards the sink reduce the amount of data forwarded to it by performing data fusion. For instance, one node can retransmit the average or the minimum of the received data only. Meanwhile, data aggregation may reduce the latency since it reduces traffic, which improves the delays. However, data aggregation techniques may worse the accuracy of the collected data. Indeed, depending on the aggregation function, original data may not be recovered by the sink, thus results in the loss of information precision.

Secondly, the adaptive sampling: The sensing tasks are energy consuming and may generate unneeded samples which affects communication resources and processing costs. Adaptive sampling techniques adjust the sampling rate at each sensor. However the application needs are met in terms of the coverage or information precision. The authors proposed the idea in [19] to adjust the acquisition frequency to the user activity because it may not be necessary to sample at the same rate when the user is sitting or running.

Thirdly, network coding (NC) is used to reduce the traffic in broadcasting scenarios by sending a linear combination of several packets instead of a copy of each packet. [20] combines the network coding and Connected Dominating Sets to further reduce energy consumption in broadcast scenarios.

Finally, data compression encodes information considering that the number of bits needed to represent the initial message is reduced. It is energy efficient because it reduces transmission times as the packet size is smaller. However, current compression algorithms are not applicable to sensor nodes because of their resource limitations. Therefore, specific techniques have been developed to adapt to the computational and power capabilities of wireless motes. Kimura and Latifi [21] have surveyed compression algorithms specifically designed for WSANs.

3) Sleep/wakeup schemes

Idle states are the major sources of energy consumption at the radio component. Sleep/wakeup schemes adapt the node activities to save energy by putting the nodes in sleep mode.

Firstly, duty cycling schemes schedule the node radio state depending on network activities in order to minimize the idle that listens and favours the sleep mode. These schemes are usually divided into three categories: on-demand, asynchronous and scheduled rendezvous [22]. Duty cycle based protocols are certainly the most energy efficient but they suffer from sleep latency because a node must wait for the receiver awake. Moreover, in some cases, it is not possible for a node to broadcast information to all of its neighbors because they are not active simultaneously. Finally, fixing parameters like listen and sleep periods, preamble length and slot time is a tricky issue because it influences network performance. For example, a low duty cycle saves a large amount of energy but increase communication delays. Thus, protocol parameters can be specified prior to deployment for simplicity, although this leads to a lack of flexibility.

Secondly, topology control: When sensors are redundantly deployed to ensure good space coverage, it is possible to deactivate some nodes while maintaining network operations and connectivity. Topology control protocols exploit redundancy to dynamically adapt the network topology, based on the specific requirement for applications in order to minimize the number of active nodes. Indeed, nodes that are not a must for ensuring connectivity or coverage can be turned off to prolong the network lifetime. Misra et al. [23] posed a solution being capable of maintaining network coverage while minimizing the energy consumption of the network by activating only a subset of nodes.

In a recent work [24], Karasabun et al. consider the problem of selecting a subset of active connected sensors based on the correlation of data. This is very useful in some applications like environmental monitoring, when the sensed data is location dependent. In this case, the data of inactive nodes can be inferred from those of active nodes due to the spatial correlation. Besides, other proposals for selecting active nodes, as [15] and [25] introduce algorithms for node selection by analysing the correlations between sensor nodes. The algorithm in [15] extracts a subset of active nodes that provide the required sensing service within error threshold, and then let the rest of the sensors go to a sleep mode. In [25] the authors proposed an application specific method for selecting sensor nodes. The proposal considers the different interests of the applications and selects sensors accordingly. However, these works do not take into account the changes occurring in the environment, which means that they might lose track of important information.

2.2.1 Self Adaptive System

There are many previous works about self adaptive system, in terms of the consideration of QoS requirements and the requirement of intelligent adjustment for example in selecting active sensor nodes. In [26], the authors describe a self-adaptive middleware for WSN. They use MAPE-K (Monitor, Analyze, Plan, Execute) feedback loops to adapt the WSN behaviour, within which the adaption is triggered whenever predefined events, considered as node context information, are detected. The adopted decision mechanism does not explicitly consider the stream nature of the produced data nether the characteristics of data distribution. Therefore, their solution is prone to cause the loss of important data or information, especially when changes happen in the environment.

Chapter 3

Basic Data Stream Processing Framework

This section provides an overview of an IoT system with support of edge computing infrastructures used in our work. The system architecture is shown in Figure 3.1. As depicted in the figure, our system contains two distinct physical tiers, namely the Things tier (left side) and the Edge tier (right side). The Things tier encompasses the IoT devices deployed in the monitored geographical area. They are responsible for collecting the physical phenomenon of interest and sending the collected data back to the Edge tier for further processing/analysis. Based on the detected changes/patterns/behavior of the collected data, Edge devices adjust the data sending rate to be used by the IoT devices. We assume that all the IoT devices have a valid communication path to reach Edge tier. Communication interference between IoT devices is not considered in this study. The Edge tier [27] encloses the more resourceful devices, e.g. smart gateways, clusters, and micro/macro datacenters. We do not make any assumption about the features of the edge nodes, but we do consider that edge nodes have no constraint on energy supply and are equipped with more storage space and powerful processing units than IoT devices.

3.1 Problem Description

The main issue to be addressed in our system_v1 is twofold, 1) how to save energy of the IoT devices by adjusting the data sending rate, which denotes the number of

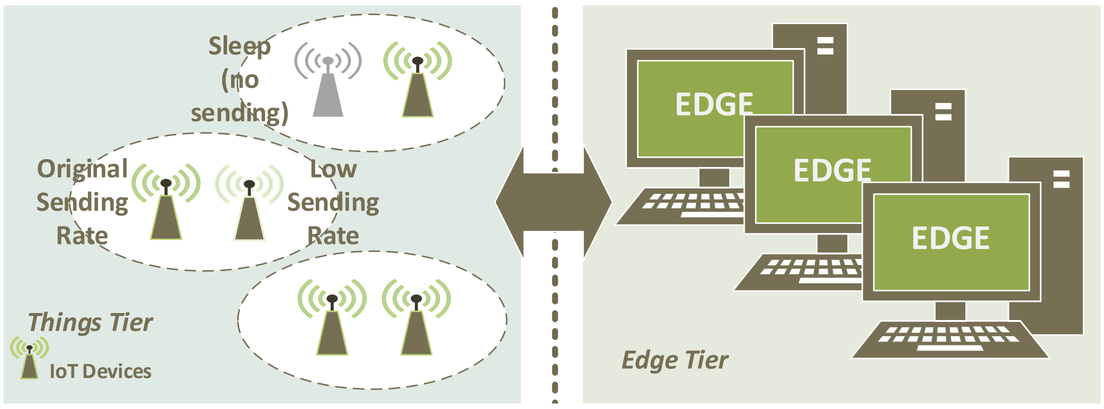


Figure 3.1: A two-tier IoT systems model

times the collected data is sent from sensor nodes to the edge node in a given time period, while still meeting the application QoS requirements, and 2) how to enable the geospatial data stream processing in an online manner, and then to aggregate the streams in a mathematical stable way for allowing further knowledge discovery.

3.2 An Energy-Efficient QoS-Aware Online Data Stream Processing Framework

In this section, we first introduce our proposed framework from a system design perspective, which indicates the design and the expectation on how the system executes the adjustment. And then we detail the system architecture and operation including the functional components of the framework, and how each component works with the others in meeting the requirements.

3.2.1 Overall system design

The overall system adjustment and control is designed based on the energy control requirement, as shown in Fig.3.1. The target control is set to be the transmission between the sensor node and the edge node.

In detail, the sensors are expected to have two active modes, active or sleep. This is an action aiming at control the transmission by turning off the transmission of selected sensors. Further more, for active sensors, there are two working modes, lower

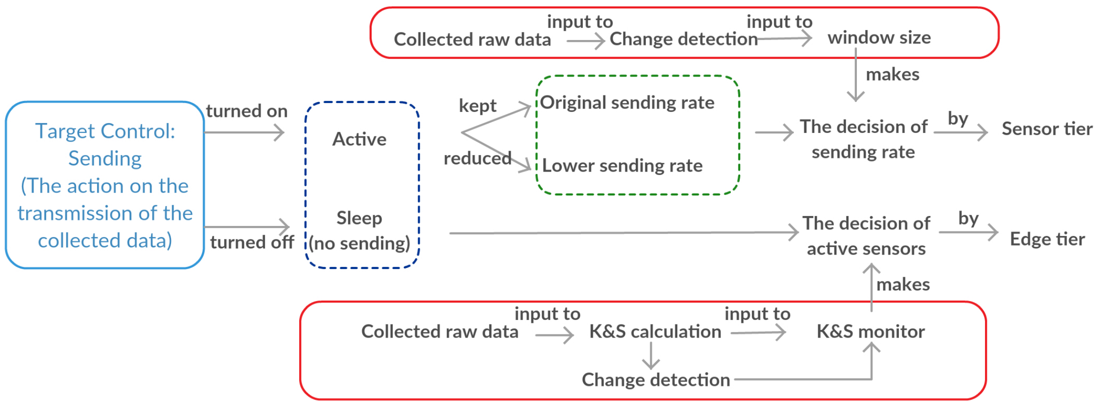


Figure 3.2: Adjustment Design

sending rate or default sending rate. By decreasing the sending rate, we further reduce the transmission from the sensors to the edge. Therefore, there are two decisions the system needs to make for adjustment, the sending rate and the active sensors. The sending rate decision is made by each sensor and the active sensors decision is made on the edge.

More precisely, the adjustment of the active sensors is triggered by the kurtosis and skewness value calculated on the edge node, and is monitored by a change detection method that aims at preventing any loss of important information caused by the reduction of active sensors. And the adjustment of the sending rate is triggered by a change detection result detected on the raw data, and is represented by a time based window. A detailed adjusting mechanism will described in the following section.

Note that, the control of transmission does not affect on the data collection, which means the sensors will work normally in collecting data, while being controlled in the sending of collected data.

3.2.2 System architecture

The main components of our proposed framework are shown in Fig.3.3. The functional components located in the sensor nodes are responsible for processing the individual streams generated by each sensing device, while the functional components located in the edge nodes are responsible to process all data streams sent from the Things tier.

In the sensor nodes, four functional components are implemented, namely **data**

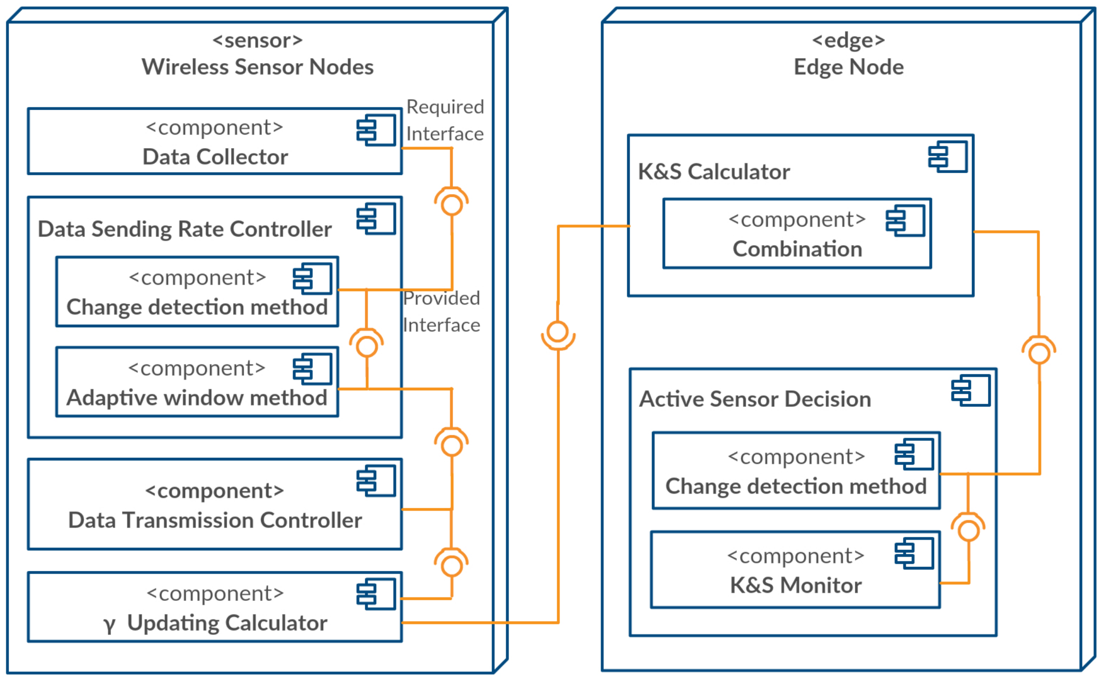


Figure 3.3: Basic System Architecture

collector, **data sending rate controller**, **data transmission controller** and γ **updating calculator**. The **data collector** is responsible for sensing the events of interest occurred in the physical world and convert them into raw data. Once raw data is collected, it will firstly be passed to the **data sending rate controller**, which contains two methods: 1) the well-known change detector and estimator ADWIN [8] adopted to serve as first step to detect changes in the distribution of the input data, and 2) the adaptive window method serves as the second step to determine the data sending rate by extracting the incoming data with a temporal sliding window. The size of the sliding window (a time-based window denoting the time interval of data sending) will dynamically control the data sending rate. ADWIN uses two data sets to detect the change: a reference data set that summarizes the past information, and a target data set sampled over the most recent input. Whenever these two data sets exhibit distinct enough means values, the algorithm concludes that the expected values within those sets are different, and a change is detected. Upon receiving a detection result, the adaptive window method calculates the new window size. If there is no change, the window size is increased, otherwise, the windows size is decreased. The new window size is then sent to the **data transmission controller** to decide the sending rate accordingly. Then under the sending rate decision, the data that is taken into account the transmission will be forward to the γ **updating calculator**. Finally, the calculated result of a sensor node will be sent to the edge node with the determined sending rate.

The edge nodes also host two functional components, namely, **kurtosis and skewness calculator** and **active sensors decision**. The data sent from the sensor nodes is the results of γ values. They are forwarded to the **kurtosis and skewness calculator** to enable combining higher order moments of multiple time-histories (denoting data sent by the different IoT devices present at the Things tier). By doing this, it offers a mechanism to reanalyze data and detect the long-term trends in the underlying process. It also has the salient features over traditional time and frequency analysis, e.g., lower sensitivity to the noise, and easy and convenient analysis of the results [28]. Afterwards, the result is used as the input for the active sensors decision component. In this component, the result will first be passed to the change detection method (ADWIN); if no change is detected, the result will be passed to the method of K&S (kurtosis and skewness) monitor. The principle of change detection on the edge nodes is the same as the one performed on sensor nodes, but the data input changes from raw data to higher order moments. This step aims at detecting the change that cannot be done at a single sensor. For example, if the change on the input does not cause any significant change on the mean value but changes where data is concentrated and that should be considered as a different phenomenon (or set of phenomena). By phenomenon, we mean an event that generates data within a specific range.

Then the $K\&S$ monitor is used to determine the number of sensors to turn off. The minimum number of active sensors that should be kept to assure the system operates properly depends on the application.

Our framework focuses on the data sending rate adjustment due to the fact that communication is the major energy consumer of sensor nodes [29]. For collecting events of interest, the sensing rate of things always remain the same at all times.

Thus the implementation of the change detection algorithm follows in two stages, one over the raw data collected by the devices and the other on the statistical moments resulting of calculating the kurtosis and skewness of collected data, thus representing the data distribution. The detection on the raw data occurs inside each sensor node, as part of the local management at the Things tier, and has the goal to save energy of devices while respecting the application QoS. The outcome of the detection process is used as an initial monitor value to trigger adaptations in the window responsible for reduction/increase/keep control of the sending rate. The detection on the two moments takes place at the Edge tier and it is used to check changes from a global view. In such step, the goal is identifying similar patterns in the data produced by different sensor nodes.

Such similarity denotes potentially redundant sensors that can be turned off for purposes of further saving energy. In order to ensure the control of accuracy, we will always need the very real data with no sampling out to monitor the accuracy, the turning off of sensors turns off the sending transmission only, with the sensors still collect data and detect change and follows the control rules inside the sensors described above. Here, a second window regulates the number of active nodes based on the result of this change detection process at the edge nodes.

3.3 Methodology and Algorithm

This section describes the methodology of our solution and the pseudo code accordingly.

3.3.1 Online stream data processing

This section explains the rationale of our online solution for data stream processing so as to enable the mathematical stable higher order moments combination at the edge node, and to avoid the loss of precision on significant digits like the naive one-pass algorithms that use the sum of powers of the samples for the estimation of central moments [30].

Mathematically speaking, the initial p^{th} order raw moment m_p and the central moment θ_p of a given set φ that composed of discrete data $x \in \varphi$, are computed as below [31]

$$m_p = \frac{1}{n} \sum_{x \in \varphi} (x)^p \tag{3.1}$$

$$\theta_p = \frac{1}{n} \sum_{x \in \varphi} (x - \mu)^p \tag{3.2}$$

where n is the sample size and μ is the sample mean. Thus, the computation of the first four statistical moments [31] is shown below:

$$\mu = \frac{1}{n}\sum_{x\in\varphi} x = m_1 \tag{3.3}$$

$$\sigma^2 = \frac{1}{n}\sum_{x\in\varphi}(x-\mu)^2 = \theta_2 \tag{3.4}$$

$$\alpha_3 = \frac{1}{n\sigma^3}\sum_{x\in\varphi}(x-\mu)^3 = \frac{\theta_3}{\sigma_3} \tag{3.5}$$

$$\alpha_4 = \frac{1}{n\sigma^4}\sum_{x\in\varphi}(x-\mu)^4 = \frac{\theta_4}{\sigma_4} \tag{3.6}$$

where μ, σ^2, α_3, and α_4 denote the mean, variance, skewness and kurtosis of the data sample, respectively, and θ_2, θ_3, θ_4 are the central moments.

The above mathematical methods of computing the statistical moments are known as the two-pass (offline) algorithm in the computing field. They are called two-pass algorithm since the mean value must first be computed in the first pass, and then the powers of the deviations from the mean are computed in the second pass, which implies the entire dataset must be retained for every computation. The computational demand regarding to one of these moments is a strong function of the number of times a sample is raised to a power. Two-pass algorithms normally require large storage space, intensive computational resource, and multiple access to the datasets. Thus, they are not suitable to be used in resource constrained devices, particularly for data stream processing.

On the contrary, one-pass (online) algorithms can be employed to tackle the aforementioned issues present in the two-pass algorithms. By one-pass algorithm, we mean that an algorithm can compute its output using only a single in-order scan of its input to avoid unbounded buffering while still guaranteeing the result can be calculated in a numerically stable fashion. A one-pass algorithm generally requires at most $\mathcal{O}(n)$ time and less than $\mathcal{O}(n)$ storage (typically, $\mathcal{O}(1)$). Thus, it is more suitable to be used on the resource constrained devices for analysing large-scale datasets on the fly.

Our solution is a robust one-pass computational statistical method, which aims at (i) computing the statistical moments without requiring the entire dataset to be processed, and (ii) enabling a straightforward combination of higher order moments from multiple data streams. This method uses variables γ_p that are easily combined by addition. After combination, the new variables are inversely transformed back to four raw moments now describing all the data, from which the statistical moments are easily calculated.

New p^{th} order moment addend variables, γ_p, are first introduced to enable straightforward combination of the statistical moments as below:

$$\gamma_p = \sum_{x \in \varphi} (x)^p \tag{3.7}$$

where $\gamma_0 = n$.

The relationship of γ_p and the corresponding raw moment m_p can be represented by:

$$\gamma_p = n m_p = \gamma_0 m_p \tag{3.8}$$

This procedure is a recursive function to calculate the latest value of γ_p for each stream. The latest value of γ_p from streams will then be combined for getting the higher order moments. So for each sensor in the sensor network, input x is obtained, and thus the gamma values are calculated whenever new data comes in. If Q sets of $\gamma_{0,q}$ are known, for $q \in (1, 2, \cdots, Q)$, then each γ_p can be represented by the equivalent p raw moments:

$$\gamma_{p,q} = \gamma_{0,q} m_{p,q} \tag{3.9}$$

One of the benefits of expressing the statistical moment by γ is that the Q sets can be readily combined by a simple addition operation, and no upper bound is set for the value of Q,

$$\gamma_{p,c} = \sum_{q=1}^{Q} \gamma_{p,q} \tag{3.10}$$

where the subscript $_c$ represents the combined γ_p.

According to this feature, for a Q sets of sensor nodes with each having input data of x, we can have their latest γ values. Thus, an online calculation of γ for individual sensor node from the Q sets is achieved, and the γ values of the whole Q sets of data can also be obtained on the fly.

These combined values from Eq.(3.10) can be transformed into raw moments by applying Eq.(3.8), then we have:

$$m_{p,c} = \frac{\gamma_{p,c}}{\gamma_{0,c}} \tag{3.11}$$

Note that $m_{p,c}$ represents the raw moments for the Q sets of x and is not necessarily equal to the sum of $m_{p,q}$. By now, we have the raw moments for the input data form all the sensors required.

According to Eq.(3.1) and Eq.(3.2), we have the well-known mathematical relationships between m_p and θ_p and thus the combined three central moments are expressed as:

$$\theta_{2,c} = m_{2,c} - m_{1,c}^2 \tag{3.12}$$

$$\theta_{3,c} = m_{3,c} - 3m_{1,c}m_{2,c} + 2m_{1,c}^3 \tag{3.13}$$

$$\theta_{4,c} = m_{4,c} - 4m_{1,c}m_{3,c} + 6m_{1,c}m_{2,c}^2 - 3m_{1,c}^4 \tag{3.14}$$

Finally, the statistical moments are computed as in Eq.(3.4) – (3.6):

$$\sigma_c^2 = \theta_{2,c}, \alpha_{3,c} = \frac{\theta_{3,c}}{\sigma_c^3}, \alpha_{4,c} = \frac{\theta_{4,c}}{\sigma_c^4} \tag{3.15}$$

The pseudo codes of the online data stream processing shown in algorithm 1, 2 respectively present how we can achieve Kurtosis and Skewness. They can be summarized as 1) updating the four intermediate values γ_{1-4}, 2) summarizing these four variables respectively using Eq.(3.10) and have the value for $\gamma_{1,c-4,c}$, 3) converting $\gamma_{1,c-4,c}$ to the first forth raw moments $m_{1,c-4,c}$ using Eq.(3.11), 4) calculating the central moments θ_{2-4} using Eq.(3.12) - Eq.(3.14), and 5) computing the Kurtosis and Skewness using Eq.(3.15).

3.3.2 Energy-efficient system behavior controllers

To achieve energy efficiency, the adaptive window technique is used to decide the data sending rate on the sensor nodes in our system. The larger the window size is defined, the lower the data sending rate is set. The pseudo code of the adaptive window is shown in Algorithm 3.

Algorithm 1 UpdateGamma

```
Input: $x, \gamma_{last}$; // input data stream; $\gamma_{last}$: previous round $\gamma$ values;
Output: $\gamma_1, \gamma_2, \gamma_3, \gamma_4$; // the calculated $\gamma$ values;
1: for each data stream do
2:     $\gamma_{1_p} = x + \gamma_{1_{last}}$;
3:     $\gamma_{2_p} = x^2 + \gamma_{2_{last}}$
4:     $\gamma_{3_p} = x^3 + \gamma_{3_{last}}$;
5:     $\gamma_{4_p} = x^4 + \gamma_{4_{last}}$
6:     $\gamma \leftarrow \{\gamma_{1_p}, \gamma_{2_p}, \gamma_{3_p}, \gamma_{4_p}\}$
7: end for
8: return $\gamma$
```

Algorithm 2 Combination

```
Input: $\gamma_1, \gamma_2, \gamma_3, \gamma_4, n_i$ //n: data sensed for each sensor
Output: Skewness, Kurtosis
1: Initialization: activeSensors ← default
2: $N \leftarrow \sum n$ // N: total number of sensed data
3: $m_1 \leftarrow \frac{\sum \gamma_1}{N}$;
4: $m_2 \leftarrow \frac{\sum \gamma_2}{N}$;
5: $m_3 \leftarrow \frac{\sum \gamma_3}{N}$;
6: $m_4 \leftarrow \frac{\sum \gamma_4}{N}$;
7: $\theta_2 \leftarrow m_2 - m_1^2$;
8: $\theta_3 \leftarrow m_3 - 3m_1m_2 + 2m_1^3$;
9: $\theta_4 \leftarrow m_3 - 4m_1m_3 + 6m_1^2m_2 - 3m_1^4$;
10: $Skewness \leftarrow \frac{c_3}{c_2^{3/2}}$;
11: $Kurtosis \leftarrow \frac{c_4}{c_2^2}$;
12: return $Skewness, Kurtosis$
```

Inspired by [11], we choose the cubic function as the utility function to control the behaviors of our adaptive window. The general form cubic function can be represented as $W = f(x^3) = ax^3 + bx^2 + cx + d$, where a is a nonzero coefficient. An inflection point separates the curve of the cubic function changing from concave to convex, and thus leads the window size W to change sharply when it is far from inflection point, and smoothly when it is close to the inflection point. To better control the change of window size, we use the following modified cubic function as our utility function.

$$W = S \times t^3 + W_{ref}, W < W_{max} \tag{3.16}$$

where S is the cubic parameter used to determine the general convergence speed, and

W_{ref} is the reference window size to decide the position of the inflection point. In addition, t is the time step, W_{max} is the maximum window size set by the different requirements of the applications.

Algorithm 3 Adaptive window

```
Input: $x$ or $Kurtosis$ and $Skewness$
Output: $W$
 1: Initialization: $W \leftarrow 0, W_{ref} \leftarrow 0, \Delta t \leftarrow 0, S, \beta, t_{step}$
 2: $change \leftarrow Adwin(input)$
 3: if change then
 4:     $W \leftarrow W_{current}(1-\beta)$
 5:     if $W_{current} < W_{ref}$ then
 6:         $W_{ref} \leftarrow \frac{W_{current}(2-\beta)}{2}$
 7:     else
 8:         $W_{ref} \leftarrow W_{current}$
 9:     end if
10: else
11:     if $W_{current} < W_{max}$ then
12:         $W \leftarrow S(t-\Delta t)^3 + W_{ref}$
13:     else
14:         $W \leftarrow W_{max}$
15:     end if
16: end if
17: return $W$
```

As mentioned before, ADWIN is used to detect changes in data streams. When a change is detected, it indicates that a potential change in the input is happening. Thus, the algorithm needs to stop decreasing the data sending rate and determines the new window size. The new W_{ref} is determined by two different scenarios. If the window size $W_{current}$ is larger than the reference window size at the time of the change occurs, W_{ref} is set to $W_{current}$, Otherwise, W_{ref} is set by the following equation:

$$W_{ref} = W\frac{2-\beta}{2} \tag{3.17}$$

where β is a decreasing factor. This action of reducing the reference window size without setting to $W_{current}$ slows down the increase of the window size afterwards. It indicates that changes are more likely to occur if the window size has not reached the reference window size. And thus efficiently prevent missing a detection of change.

In summary, the adaptive window works as below:

$$W = \begin{cases} S(t-\Delta t)^3 + W_{ref}, & (nochange, W < W_{max}) \\ W_{max}, & (nochange, W \geq W_{max}) \\ W_{current}(1-\beta), & change \end{cases} \tag{3.18}$$

where $t = 0, t_{step}, 2t_{step}, ...$ or equal to the natural elapsed time, δt is the time period that the above function takes to increase $W_{current}$ to W_{ref} when no further change is detected and it is calculated by the following equation:

$$\Delta(t) = \sqrt[3]{\frac{W_{ref} - W}{C}} \tag{3.19}$$

At the edge node, the output of ADWIN is used as the input for K&S monitor to decide the number of active sensors. The pseudo code of K&S monitor is as below.

Algorithm 4 Active Sensors

```
Input: $Kurtosis$ and $Skewness$
Output: The Number of Active Sensors
 1: if (!Adwin(input)) then
 2:     $numActiveSensors \leftarrow$ K&S MONITOR(input)
 3:     // if no change, the Active Sensors update decided by the K&S Monitor
 4: else
 5:     Reset the Number of Active Sensors, and recalculate Kurtosis and Skewness
 6: end if
 7:
 8: function K&SMONITOR($kurtosis$, $skewness$)
 9:     if ($|skewness| > S_1$ then
10:         if $kurtosis \leq K_1$ then
11:             $numActiveSensors \leftarrow Value1$ //New active sensors number
12:         else
13:             $numActiceSensors \leftarrow Value2$
14:         end if
15:     else
16:         $numActiveSensors \leftarrow Value3$
17:     end if
18: end function
19: return $numActiveSensors$
```

The values of kurtosis and skewness mostly reveal the phenomenon situation on the monitored area. Based on the values, it is possible to have a rough idea about the states of the monitored area so as to decide the number of active sensors accordingly. For example, on one hand, we can have the case that all sensors are collecting data related to a single phenomenon and, thus, it is possible to decide to keep a small number of active sensors. On the other hand, we can have the extreme case where all sensors are collecting data related to different phenomena, and thus, it is not possible to turn off any sensor.

To determine phenomena, we will first use skewness. If the data stream has a moderate skew, this means that there are one or more phenomena on the monitored area with very different data ranges. In this case, the multiple phenomena can be easily identified. Thus, the number of active sensors can be kept low but the sensors cannot be turned off all at once since it would change the kurtosis and skewness. Unlike the moderate skew case, a strongly skewed data stream indicates that multiple phenomena are taking place on the monitored area with close or overlapping data ranges. The distinction among phenomena with more or less concentrated overlapping is provided by the kurtosis index.

For given kurtosis, the shape of the peak can be used to indicate the number of occurrences of a certain situation. To better identify them, we utilize the concepts of platykurtic and leptokurtic from [31], which platykurtic means the data with a negative exceed kurtosis compared to the normal distribution, and the opposite for leptokurtic. Thus, the more often a certain situation occurs, the more leptokurtic is the data stream (which means that the more a certain data is generated, the greater is its concentration). Note that the more precise threshold to classify the kurtosis can be application specified.

A leptokurtic data stream indicates that the data are more concentrated. If not a single phenomenon is occurring, there are multiple similar phenomena that share close or overlapped data range, and even at the same time sharing a similar concentrated area. In this situation it is harder to differentiate the phenomena. Therefore, the adjustment of active sensors should be treated cautiously, which means that the number of active sensors should be kept high. A platykurtic data stream indicates different phenomena with close data ranges, but concentrates at different data ranges. This indicates that the data distributes more evenly. In this case, it is easier to differentiate the phenomena than the laptokurtic ones. The number of active sensors can thus be kept lower.

The classification result is then used to decide how the system adjusts the number of active sensors. And this is decided by the K&S Monitor which is, as mentioned above, triggered by the change detection result on the edge node. Note that in the current version of our algorithm, we simply set the number of active sensors to be a fixed value instead of dynamically change it. Under the decision of the number, the system randomly selects this number of sensors from the original set of sensors.

3.4 Performance Evaluation

We wrote our own simulator in Java language to implement the proposed framework and evaluate its performance. We performed the experiment considering 200 sensor nodes geographically deployed and two Smart Grid applications used in [10]. One application is the overhead power line monitoring and the other is battery monitoring used on the transmission towers. A transmission tower is a tower used to support an overhead power line. An overhead power line is an electric power transmission line suspended by towers. The overhead power-lines are the most common mean to transmit energy. In the Smart Grid context, the transmission tower is also responsible for storing energy using local batteries. The main goal of the performed experiments is to evaluate 1) the adaption mechanism in terms of its correctness and its responsiveness, including the details how the system has adapted itself to, how the change detectors work and whether the system response is correctly and timely regarding to the applications, and 2) the system performance in terms of the accuracy (expressed by the error rate of the system outputs to the predefined real cases) and energy saving (expressed by the energy consumption rate to the initial consumption without adjustment).

In our experiment, two applications share the same information (temperature), but with different data ranges, and are yet correlated. For evaluating the system performance, we set up four time slots, namely, T1, T2, T3 and T4. Each time slot represents a particular state of the applications along the time, denoting a particular event to be detected. T1 represents ideal conditions for both applications (a safe condition). T2–T4 represents an overheated scenario for the overhead power line, the battery temperature or both. We conducted three experiments and each of them contains a transformation from the ideal case to a worse one, e.g., T1$\rightarrow$T2, T1$\rightarrow$T3 and T1$\rightarrow$T4.

In T1, applications generated a highly skewed but platykurtic data stream. In T2, applications generated a highly skewed but leptokurtic data stream. In T3 and T4, moderate skewed data streams were generated. In our experiments, 100 sensors are used for the battery and another 100 sensors for the overhead power line. By default, all sensors collected data and sent to the edge node in every 15 seconds, as suggested by

[10]. Battery and overhead power line temperatures changed according to the thermal models presented in [32]. The thresholds used in in the K&S Monitor for classifying the Skewness and Kurtosis are set to $|0.5|$ and 0 (the exceed Kurtosis value). All data points are averaged over 500 simulation rounds, and each round lasts 120 hours.

	Temperature Range			
	T1	T2	T3	T4
Battery	40 °C-65 °C	65 °C-75 °C	40 °C-65 °C	65 °C-75 °C
Power Line	40 °C-144 °C	40 °C-144 °C	144 °C+	144 °C+

Table 3.1: Temperature Range

Situations	Avg.Sending Rate	K&SMonitor output Number of Active Sensors			Transmitted Data	Error Rate
		Initial	Situation1	Situation2		
T1→T2	30 s	200	100 (T1)	140(T2)	15%	1.5%
T1→T3	30 s	200	100 (T1)	80 (T3)	15%	1.2%
T1→T4	30 s	200	100 (T1)	80 (T4)	14%	0.4%

Table 3.2: System Adjustment Output (Transmitted Data is the total percentage of transmitted data comparing to the number of collected data)

Situations	CDDT: SN		Avg.CDER: SN	CDDT: EN	CDER: EN
	Max	Avg.			
T1→T2	95 s	47 s	0.04%	17 min	0.25%
T1→T3	240 s	63 s	0.02%	1 min	0.26%
T1→T4	45 s	37 s	0.02%	20 min	0.29%

Situations	K&SMonitor Error Output			
	Situation 1		Situation 2	
	Length	End	Length	End
T1→T2	64 min	146 min	38 min	221 min
T1→T3	73 min	151 min	37 min	203 min
T1→T4	70 min	160 min	22 min	138 min

Table 3.3: Change Detection and K&S Monitor Performance (CDDT: change detection delay time, CDER: change detection error rate, SN: sensor node, EN: edge node)

3.4.1 Evaluate the performance of adaptation mechanism

We firstly evaluate how the system adapts itself accordingly to the predefined changes which are represented by significant drifts in the data values that move the distributions to new states. Table 2 summarizes the results of the adjusted sending rate (15s→30s) and the number of active sensors. Table 3 analyzes the change detection on the sensor and the edge node, in respect to the detection delay and detection error. The ”K&S Monitor Error Output” in Table 3 shows the time period of the incorrect classification decision last and when it ends. They always happened at the beginning of the experiment while the amount of input data is insufficient so that leads to a misclassification. However, these mis-classifications end up very shortly once our system has enough input. According to these two tables, our system adapts well to the changes happening in the system.

3.4.2 Evaluate the change of the adaptive window

In this experiment, we specifically demonstrate how the window size is increased/decreased according to the predefined changes. For illustration purpose, we present how the window size (time interval of data sending) changes during the run time for two randomly selected sensors as example. Fig.3.4 clearly shows how the adaptive window method performs on these two sensors during a long period of time. Also, we selected four critical time points (A, B, C, and D) to further explain how the sensors behave. At time A, the system adjusted the number of active sensors when a change was detected and the sensor No.1 was turned off accordingly. At time B, a change was detected on the sensor No.32, so the window size was reduced. Then from B to C, a clear window size growth curve was occurred after a change was detected. Finally, at time D, a change was detected on the edge node and thus triggered an adjustment on the number of active sensors. This resulted in an activation of sensor No.1, and a turning off of sensor No.32.

3.4.3 Evaluate the energy saving with accuracy

Finally, we studied how the change of the amount of processed data affects the data accuracy and the energy consumption. To do so, we compared these values when using our proposal to the reference original produced data stream (without the controlling of the sending rate and active nodes). We implemented the energy model presented in [33], and simulated the energy consumption of 1) sensing, 2) sensor logging, 3) data transmission and 4) data processing. We measured the data accuracy by using the

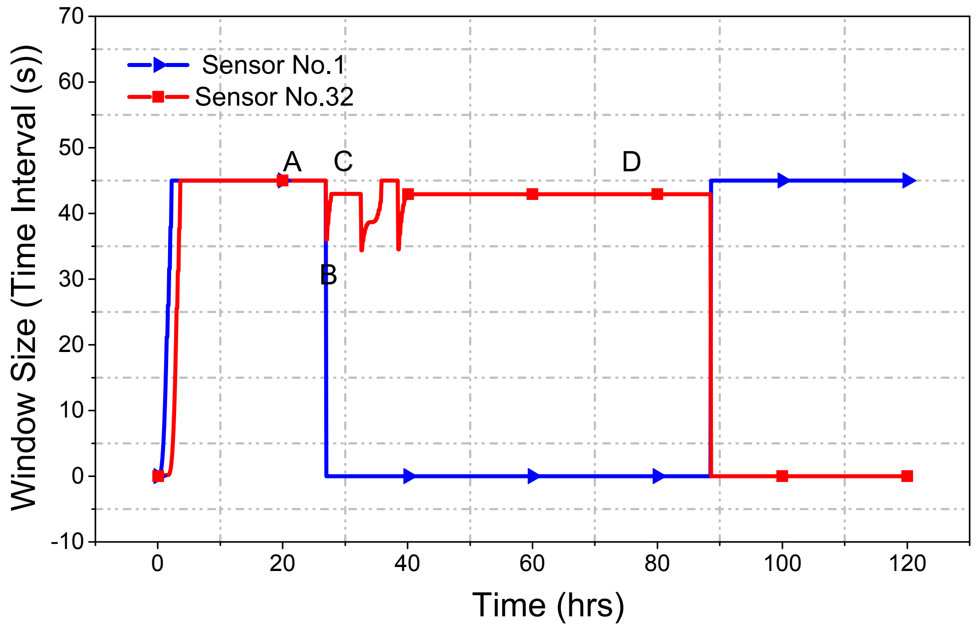


Figure 3.4: The change of window size for two sample sensors

metric of error rate, which is defined as the results of kurtosis and skewness computed by our system, compared to the result computed by the two-pass algorithms using the full original data stream. The results of data accuracy and energy saving are shown in Fig.3.5. From the figure, we notice that: 1) the reduction of the total amount of processed data decreases the energy consumption but increases the error rate, which means loss of accuracy, 2) We achieve a significant energy saving with certain tolerance of error rate. The best case in our simulation consumed almost only 0.3% of the initial energy, with the worst error rate about 5.5%. And upon the best accuracy with an error rate of 0.034% (kurtosis) and 0.068% (skewness), the system consumed nearly just half of the energy consumption of the benchmark.

3.5 Conclusion

In this paper, we presented an energy-efficient QoS-aware online stream data processing on Internet of Things systems. With Edge tier, our system can dynamically control the energy consumption to meet the accuracy requirement of different applications. The simulation results show that our proposed framework is promising. In the future, we plan to evaluate our framework on real sensor nodes. Another possible extension

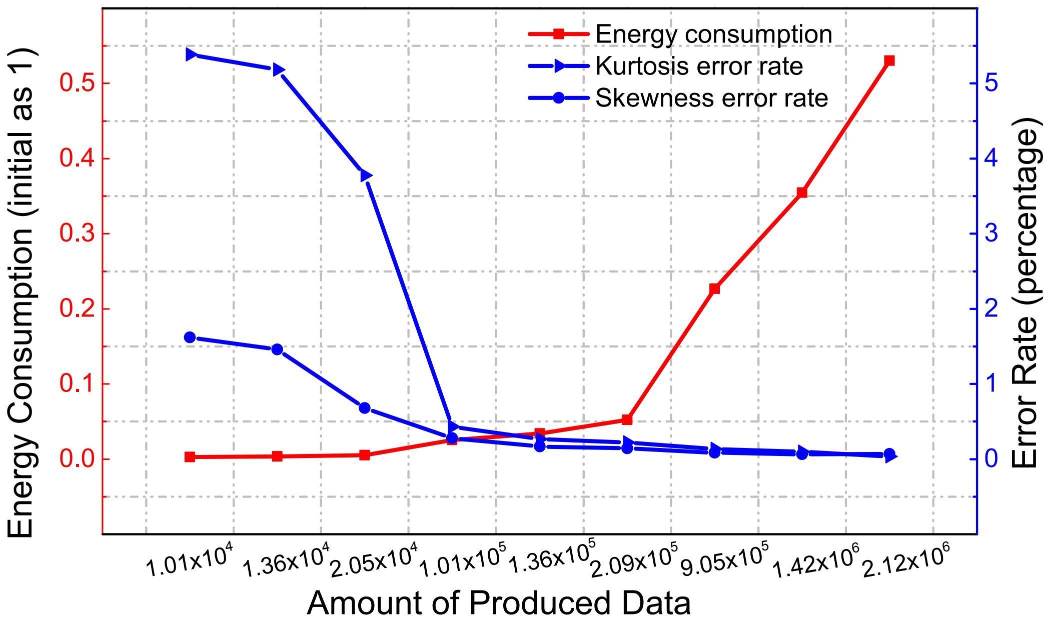


Figure 3.5: Energy consumption with error rate

is to evaluate in a scenario where the edge nodes can collaborate among themselves. Finally, we intend to extend the algorithm to work in the scenarios where both the number of active sensor nodes and the data transmission rates of the nodes can change dynamically.

Chapter 4

Enhanced Data Stream Processing Framework

To better solve the problem stated in Chapter 3, we further adjust the system functional architecture and the control mechanism on the edge node, aiming at better saving energy and better controlling the active sensors.

Two main improvements based on the previous system are: 1) the transmission information from the sensors to the edge, 2) the selection mechanism on the active sensors. Details on the new version of framework is addressed in **Section 4.2**, followed by and overall evaluation on the system and the comparison of the two versions of framework.

4.1 Problem Description

This section provides an overview of an IoT system with support of edge computing infrastructures used in our work. The system modeling context is the same as before as shown in Fig.3.1.

In the previous system version(system1), we calculate γ values inside the sensor node and send the values to the edge. As described in Algorithm 1, each input data has four γ representing the first four orders of intermediate values of calculating the statistical moments. So each transmission contains four data values. In this case, each input data results in four values for transmission, which highly increase the transmission load comparing to the transmission for just the input data.

Secondly, in system1, the selection of active sensors is based on a rough classification on the distribution. The overall kurtosis and skewness calculated for all the transmitted data roughly indicates the possible real situation, complex multiple similar phenomena, or less complex multiple different phenomena, or simple single phenomenon. Therefore, the selection based on such classification might just make it less possible to lose information. And we tested the adjustment based on a situation of simple combination of phenomena, and used experimental safety number of active sensors. In this case, we are not able to see how system1 works with more complex situations.

Considering the drawbacks and limitations of system1 stated above, we intend to improve the overall system. Therefore, we proposed an enhanced version of system, namely system2, under the same issue addressed previously, which is to consider the need of the effective processing of data streams and the efficient usage of the available resources of devices. The two key points considered as the enhancement in system2 are: 1) how to save more energy of the IoT devices without losing the accuracy comparing to system1, and 2) how to select active sensor nodes with a more intelligent mechanism, to understand and reveal the situation more precisely and to be able to work with more complex situations.

Therefore, the targeting improvements on the system are in two folds in regard to the enhancement requirements, 1) the change on the data sent form the sensor node to the edge, 2) the change on the method of selecting active sensors. Details of the improvements to system are:

1) We simplify this transmission by moving the calculation of γ values to the edge. The transmission between the sensor node and the edge node will contain the single raw data only. This will decrease the number of transmitted data by the sensors in system 2 to a quarter of which in system1, when processing the same number of data.

2) We change the nodes selection method to a clustering based mechanism. This enables a better understanding on the situation, especially with multiple phenomena. Then the selection of active sensors is based on the clusters. This better prevents turning off the wrong sensors that are important in representing specific phenomena. For example, if there is just a single sensor that works on a specific phenomenon, it is very possible that system1 turns off this sensor. Hence, applying the clustering mechanism can prevent such loss by assigning this sensor to an individual cluster.

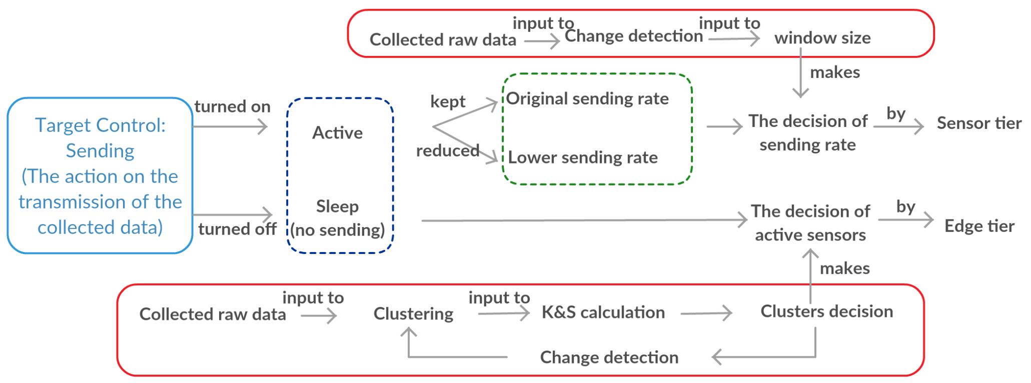


Figure 4.1: Adjustment Design

4.2 An Energy-Efficient QoS-Aware Online Data Stream Processing Framework

In this section, we first introduce our proposed framework from a system design perspective, and then we detail the functional components, both with the comparison to the previous one.

4.2.1 Overall system design

Targeting on the intended improvements, we make change to the design of the system adjustment and control, on the decision of the active sensors made on the edge node changes. As stated above, it will be improved to a clustering-based selection method.

The general idea on the design of sensors active mode remains the same. Adjustment applies on the decision of active sensors made on the edge, as shown in Fig.4.1. The sensors are clustered into certain clusters based on the raw data. And the active sensors are decided inside each cluster. A change detection method is used to monitor the change and trigger the re-clustering of the sensors.

4.2.2 System architecture

Following the new system requirements, the system changes the architecture on 1) the γ updating calculation that is moved from the sensor node to the edge, and 2) the functional components on the edge nodes.

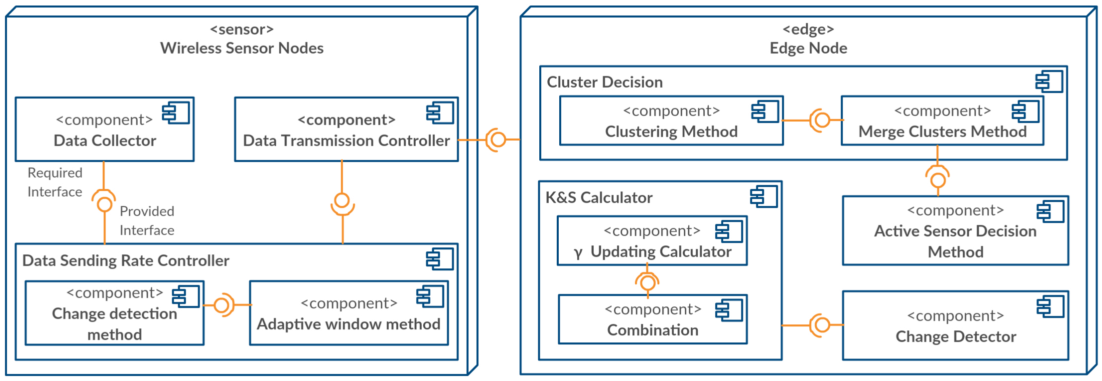


Figure 4.2: System Architecture

As shown in Fig.4.2, three functional components implemented in the sensor node, namely **data collector**, **data sending rate controller** and **data transmission controller**. Same as the previous system, the data collector is responsible for sensing the events of interest that happened in the physical world and convert them into raw data. Once raw data is collected, it will be passed to the data sending rate controller, which contains two methods: 1) the well-known change detector and estimator ADWIN that is adopted to serve as the first step to detect changes in the distribution of the input data, and 2) the adaptive window method serves as the second step to determine the data sending rate by extracting the incoming data with a temporal sliding window. This is a time-based window of which the size represents the time interval of two data to be sent to the edge. So the window size is calculated by the adaptive window method and is used to control the sending rate. That is to say, under certain sending rate that decides the data sending time interval, only the data that collected later than the current time interval will be sent to the edge. But all the collected data will be input to the change detector to monitor the input data.

ADWIN uses two data sets to detect the change: a reference data set that summarizes the past information, and a target data set sampled over the most recent input. Whenever these two data sets exhibit distinct enough means values, the algorithm concludes that the expected values within those sets are different, and a change is detected. Upon receiving a detection result, the adaptive window method calculates the new window size. If there is no change, the window size is increased, otherwise, the window's size is decreased. The new window size is then sent to the data transmission controller to decide the sending rate accordingly.

Finally, the processed data stream of a sensor node will be sent to the edge node with the determined sending rate. At this time, γ values will not be calculated in the sensor node.

The edge node hosts four functional components, namely, **cluster decision**, **K&S (kurtosis and skewness) Calculator**, **change detector** and **active sensors decision**. To demonstrate how the edge node works under these four components, we divide the work flow into two stages, the initialization stage and the working stage. When data is firstly sent from the sensor nodes, it is firstly received and processed by the Cluster Method of Clustering Decision component. And then the clusters result is forwarded to the K&S Calculator. Thus, with receiving the clusters result and the continuously coming data, the K&S Calculator matched the coming data with the clusters and calculate the kurtosis and skewness for each cluster. The kurtosis and skewness values for each cluster are then being passed to the Merge Clusters Method to decide the final clusters, by deciding whether there are clusters being able to be merged to one cluster. Finally, the clusters result is processed to the Active Sensor Decision component and is used to decide the active sensors. By now, the initialization stage is completed.

After the initialization, the edge node works into a normal working stage. In this stage, the coming data will no longer be sent to the Cluster Decision component. Instead, new coming data are passed directly to the K&S Calculator to get the kurtosis and skewness results. Then the kurtosis and skewness values are passed to the Merge Clusters Method, as well as the Change Detector. If the clusters are merged, new clusters will be passed to the Active Sensors Decision again to update the active sensors. For the Change Detector, if there is change in the kurtosis and skewness values, the system will reset all the adjustment result and start over from the initialization stage.

The principle of change detection on the edge nodes is the same as the one performed on sensor nodes, but the data input changes from raw data to higher order moments. This step aims at detecting the change that cannot be done at a single sensor. The minimum number of active sensors that should be kept to assure the system operates properly depends on the application. Also note that the index of same of the kurtosis and skewness is application specified in terms of phenomena and precision level.

Note that, 1) there are thresholds defined in the initialization. They are application specific. For example, the threshold when the kurtosis and skewness are regarded stable and being passed to the Merge Clusters Method. 2) Turning off the sensors means only turning off the transmission. All the sensors will remain collecting data, detecting change inside the sensor node and adjusting the sending rate. Therefore, when there is change detected inside the sensor, it will re-activate by itself. The edge will start receiving the data from this sensor. This action reduces the possibility of losing important information because of the active sensors adjustment. But the change that occurs in the sensors are not necessarily resulting in the change of phenomenon. So, the edge node will not reset the adjustment unless change is detected in the edge node too.

4.3 Methodology and Algorithm

This section describes the new methodology, clustering, of our new solution and the pseudo code accordingly.

4.3.1 Online stream data K-means based clustering

In our system, the initial clustering is based on K- means method.

Traditional K-means method is a process of updating the centers for each class. And then based on the distance for each data point to the centers, K-means aiming at and assigning data points a classification to the class where they have the minimum distances to the center. In this case, the updating of the new center will be the mean value of all the data points currently belong to the class. After a while, when no data point change its class, the classification is done.

However, a same problem as the calculation of the kurtosis and Skewness occurs. The traditional K-means requires a whole data set. The updating of the mean value requires the exact number of the data points. While we are dealing with data stream, which is an online processing. Old data will not be stored in the system. We will not be able to calculate the mean value after the first classification which is usually based on a group of random centers. Thus we need an online K-means. Inspired by [34], we change the center updating mechanism. We are not calculating the mean values anymore, we set a fixed weight with every updating that makes it an uncounted updating rule. In detail, the traditional K-means calculates its center using:

$$c_i = \frac{\sum x_i}{n_i} \tag{4.1}$$

Thus, in an online context, data is one by one received by the system. In this case, upon receiving each new x_i to class c_i, the number of data point in this class will be n_i, and the center will be updated using:

$$c_i\text{'} = c_i + \frac{x_i - c_i}{n_i} \tag{4.2}$$

So the online K-means aims at introducing $\alpha \in (0, 1)$ to replace $\frac{1}{n_i}$, and hence the center updating method with receiving each data point that belongs to class will be:

$$c_i\text{'} = c_i + \frac{x_i - c_i}{\alpha} \tag{4.3}$$

Therefore, the overall clustering procedure is 1) get new data point, 2) determine the closest center to the input, 3) update the center of the cluster with new assigned data.

On the bases of the above clustering, our system implements a further refinement of the clustering, aiming at minimizing the number of clusters to maximize the energy saving. The idea is to merge the clusters that are not necessary to be differentiated. For example, a phenomenon with certain distribution is separated because the clustering decision is made before a long term run. The initial K-means clustering used the mean value to present the cluster. But this cannot always represents the phenomena, as described in the previous system, higher order moments are in need. We use kurtosis and skewness as well to refine the cluster based on the data distribution. Two clusters are merged if they process equivalent kurtosis and skewness value [35].

The pseudo code is shown in Algorithm 5:

Note that, in our system, only the clusters that detected changes will be reset. This means that only the sensors that belong to a cluster with change will be re-clustering.

4.4 Performance Evaluation

We wrote our own simulator in Java language to implement the framework previously described and evaluate its performance. We performed the experiment with 200 sensor nodes that are geographically deployed and two Smart Grid applications used in [10]. One application is the overhead power line monitoring and the other is battery monitoring used on the transmission towers. Our goal is to evaluate 1) the adaption

Cases	Status	
	status1	status2
1	T1 + T2	T1
2	T1	T1
	T1	T2
	T1	T3

Table 4.1: Experiment Cases

mechanism and 2) the system performance in terms of the accuracy and energy saving. 3) the performance comparison of the two versions of system.

We have chosen as experimentation scenario the same as the previous version. A transmission tower is a tower used to support an overhead power line. An overhead power line is an electric power transmission line suspended by towers. The overhead power-lines are the most common mean to transmit energy. In the Smart Grid context, the transmission tower is also responsible for storing energy using local batteries.

In our experiment, two applications share the same information (temperature), but with different data ranges, and are yet correlated. For evaluating the system performance and to make a comparison to the previous system version, we set up the same four time slots, namely, T1, T2, T3 and T4. Each time slot represents a particular state S1, S2, S3, S4 of the applications along the time, a particular event to be detected. T1 represents ideal conditions for both applications (a safe condition). T2–T4 represents an overheated scenario for the overhead power line, the battery temperature or both. We conducted the experiment on the case, 1) from a status that starts with a combination of situations to a single situation. This is to evaluate how the system works on the function design. and 2) the same three experiments with each of them containing a transformation from the ideal case to a worse one, e.g., T1$\rightarrow$T2, T1$\rightarrow$T3 and T1$\rightarrow$T4. This is to see the performance comparing to the previous version. The two cases are presented in Table 4.1.

More detail, for the first case, we assume the combination means a complex situation in the beginning. And after a period of time, it tends to be a stable and simple situation. Initially, three sensors on the battery are in the state of T2($65\,^{\circ}\mathrm{C}$ $75\,^{\circ}\mathrm{C}$) and the others T1($40\,^{\circ}\mathrm{C}$ $65\,^{\circ}\mathrm{C}$). And three sensors on the overhead power line are in the state of T3($144\,^{\circ}\mathrm{C}+$) and the others T1($40\,^{\circ}\mathrm{C}$ $144\,^{\circ}\mathrm{C}$). After a while, all sensors are in the state of T1. All data points are averaged over 500 simulation rounds, and each round lasts 120 hours.

4.4.1 Evaluate the clustering mechanism

We firstly evaluate the clustering mechanism without sending rate and active sensors adjustment, in order to see if the system can achieve the required clusters.

In order to do this, we conduct the experiment in the context of the first case. So based on the case setting, there are four clusters in the beginning and after a period of time there turns to two clusters. There should be an intermediate stage when change occurs and the two clusters containing the sensors that change to T1, will be reset.

The testing result is shown in Fig.4.3. Fig.4.3(a) indicates the predefined situation. Fig.4.3(b) presents the initial clusters in time point t1. There are four clusters with ten sensors each of the battery and the overhead power line different from the others. And at time point t2, as shown in Fig.4.3(c), two clusters changed but haven't merged yet. So ten sensors of each changed the cluster but the number of clusters remain four. Then at t3, as shown in Fig.4.3(d), merging mechanism was activated and resulted in the final clusters with only two clusters. One of the clusters represents the battery sensors and another the overhead power line sensors. According to the result, our clustering mechanism works well in achieving the expected clustering result.

4.4.2 Evaluate the performance of adaptation mechanism

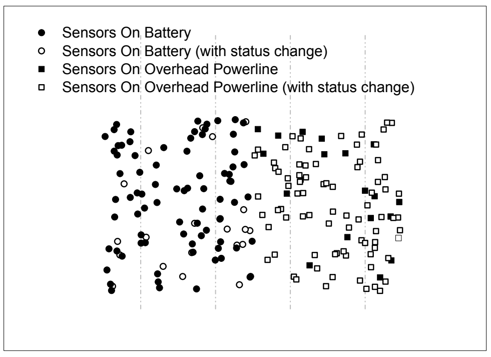
We then evaluate the system performance on its adaptation mechanism, under the context of case1. With the setting of case1, there is one change that occurs on the 20 sensors that change the state. And there is one change on the edge that activate the re-cluster for 2 clusters of the 4 initial clusters. And the maximum sending rate is set to be 45 seconds.

The testing result is presented in Table 4.2 and Table 4.3.

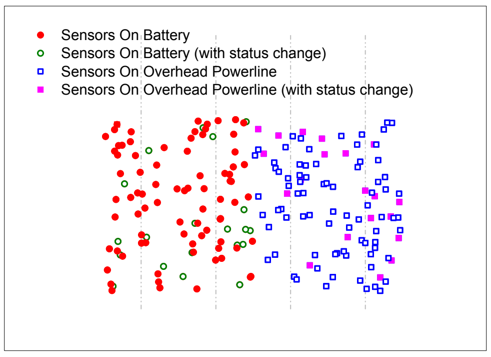
Time Slot	Avg.Sending Rate	Number of Active Nensors	Total Percentage of Transmitted Data	Error Rate
t1	40 s	4	20.6%	3.47%
t2	20 s	200	16.9%	1.31%
t3	43 s	2	3.13%	0.08%

Table 4.2: System Adjustment Output (Error Rate: the error rate in terms of the kurtosis and skewness for the whole data stream at the end time point of the observation)

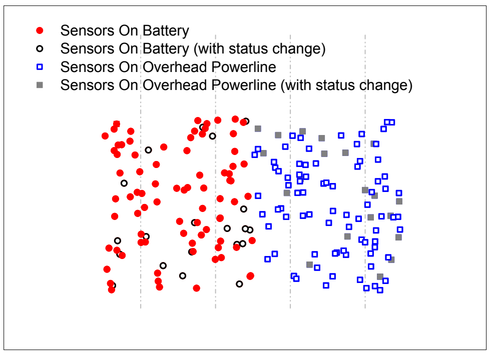
Table 4.2 presents the adjusting result at three time slots, t1, t2 and t3. Time slot t1 is the first observation time point before predefined change occurring time. Time slot



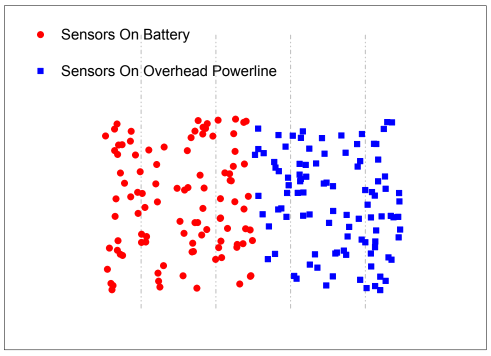
(a) Pre-defined clusters



(b) Clustering Result at t1



(c) Clustering Result at t2



(d) Clustering Result at t3

Figure 4.3: Clusters

t2 is the second observation time point after change occurs but before the re-cluster being done. Time slot t3 is the last time point of the experiment.

From Table 4.2, the average sending rate tend to reach the maximum except for the change detected period, with a significant decrease when change occurs. And the number of sensors are set as expected, with acceptable error rate.

Table 4.3 analyzes the change detection and the clustering from the testing result of three example tests. There are delay on the change detection and clustering mainly because: 1) the detection and clustering mechanism has its own comparing depository. This can be adjust by applying different detector based on different precision requirements. 2) When change occurs, the input data tends to be unstable. Thus it requires a certain time period for the system to recognize the situation.

Tests	CDDT: SN		Avg.CDER: SN	CDDT: EN	CDER: EN	Avg. Clustering Output Delay
	Max.	Avg.				
test1	90 s	42 s	0.02%	23 min	0.23%	57 min
test2	102 s	50 s	0.03%	17 min	0.20%	60 min
test3	95 s	47 s	0.02%	18 min	0.18%	55 min

Table 4.3: Change Detection and Clustering Performance (CDDT: change detection delay time, CDER: change detection error rate, SN: sensor node, EN: edge node)

4.4.3 Evaluate the energy saving with accuracy

Finally, we studied how the change of the amount of processed data affects the data accuracy and the energy consumption. To do so, we compared these values when using our proposal to the reference original produced data stream (without the controlling of the sending rate and active nodes). We implemented the energy model presented in [33], and simulated the energy consumption of 1) sensing, 2) sensor logging, 3) data transmission and 4) processing in our work. We measured the data accuracy by using the metric of error rate, which is defined as the results of kurtosis and skewness computed by our system, compared to the result computed by the two-pass algorithms using the full original data stream.

By adjusting the setting of the number of active sensors in each cluster, we achieve different amount of energy saving. And the results of data accuracy with the increase of energy saving, of random selected two example tests are shown in Fig.4.4. From the figure, we observed that the error rate are kept satisfied below 0.6%. Thus we are able to achieve a significant energy saving with certain tolerance of error rate. The most energy saving (the minimum energy consumption) that happened when we keep one sensor for each cluster. In this case, we have the error rate at around 0.4%.

4.4.4 Evaluate the improvement

Finally, we compared the two system versions. Firstly, we compare the adjustment of the two systems: 1) compare under case2 in the three time slots, t1, t2 and t3 as above. 2) compare under case1 and presented the average result for the three situations, T1⇒T2, T1⇒T3 and T1⇒T3. The comparison results are shown in Table 4.4 and Table 4.5. Secondly, we presented the comparison of the two versions of the system in regard to the energy consumption with error rate. We presented the energy

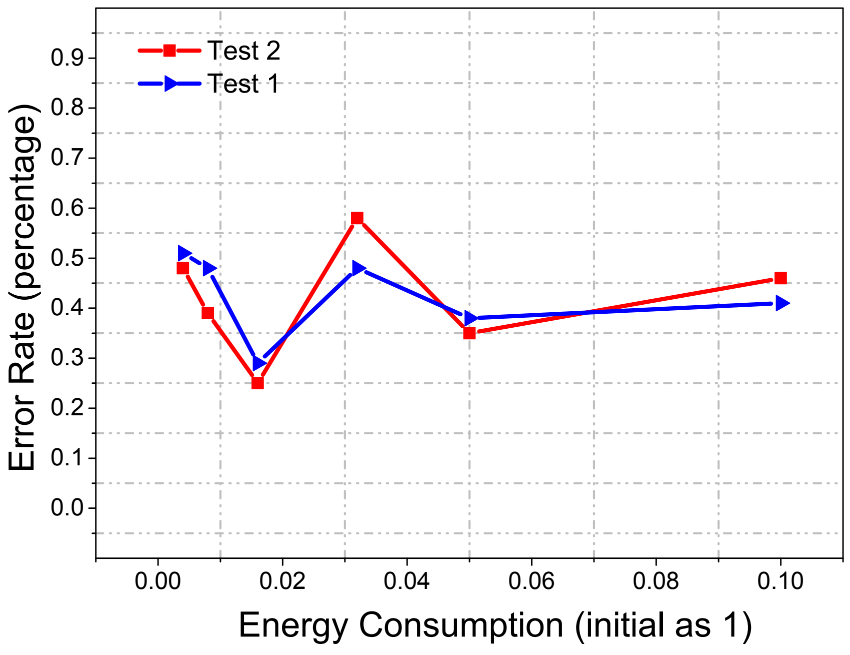


Figure 4.4: Energy consumption with error rate

consumption for both systems under accuracy. The consumption value is an average consumption under specific accuracy in all the experiments.The comparison results are shown in Fig.4.7.

4.4.4.1 Compare the system adjustment

Time Slot	Average Sending Rate		Number of Active Sensors		Total Percentage of Transmitted Data		Error Rate	
	S1	S2	S1	S2	S1	S2	S1	S2
t1	40 s	40	140	4	26.8%	20.6%	56.5%	3.47%
t2	30 s	20	140	200	26.1%	16.9%	37.8%	1.31%
t3	40 s	43	120	2	18.2%	3.1%	30.5%	0.08%

Table 4.4: Comparison of System Adjustment Output in Case2

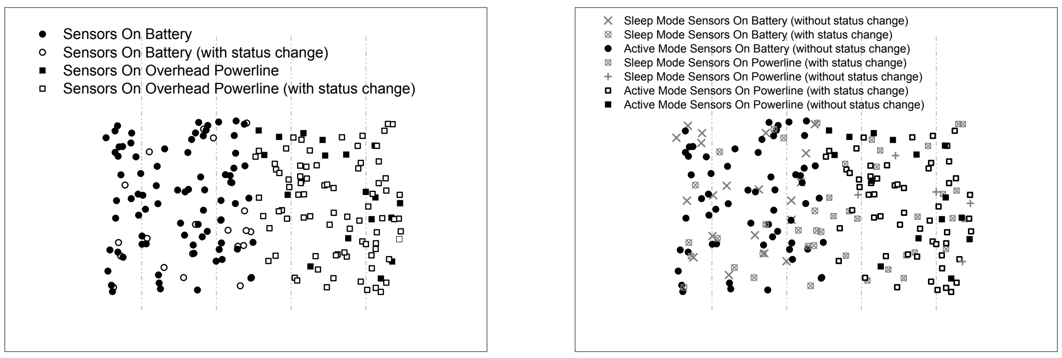
From Table 4.4, we can see that in case2, a more complex situation (the initial is a combination of several situations), the previous system is not able to differentiate several situations in the beginning. The system regard it simple situation and decide

Situations	Average SR		Number of Active Sensors				Transmitted Data (%)		Error Rate	
	S1	S2	S1		S2		S1	S2	S1	S2
			Sit1	Sit2	Sit1	Sit2				
T1→T2	40 s	35	100	140	2	2	13%	1.0%	0.8%	0.15%
T1→T3	40 s	35	100	80	2	2	13%	0.9%	1.4%	0.15%
T1→T4	40 s	35	100	80	2	2	12%	1.0%	0.7%	0.09%

Table 4.5: Comparison of System Adjustment Output in Case1 (Transmitted Data: Total Percentage of Transmitted Data)

the active sensors based on the kurtosis and skewness. Fig.4.5 and Fig.4.6 show the example test results presenting the active sensors selected by the two systems in the three time slots. The selection of the system1 shows that the system failed to keep several necessary sensors on. For example, in t1, as shown in Fig.4.5(b), all the sensors on the battery that represent a different situation were turned off. This results in a significant error rate in the result. But in t3, the situation turned to be simple without combination (T1), the error rate decreases. In contrast, as shown in Fig.4.6(b), Fig.4.6(c) and Fig.4.6(d) the selection of active sensor nodes is in accordance with the predefined situation. In this case, the new system performs a lot better than the previous one.

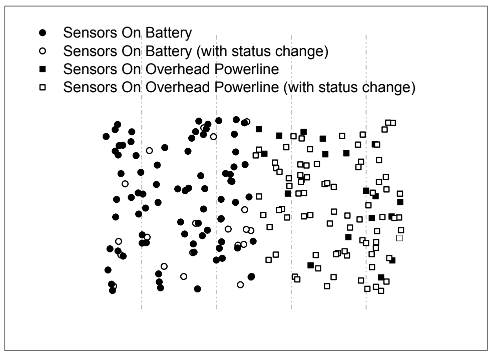
From Table 4.5, we can observe that under case1 which is simple situation, both of the two versions of system perform well. While the new system keeps less active sensors and transmits less data, which can achieve better energy saving.



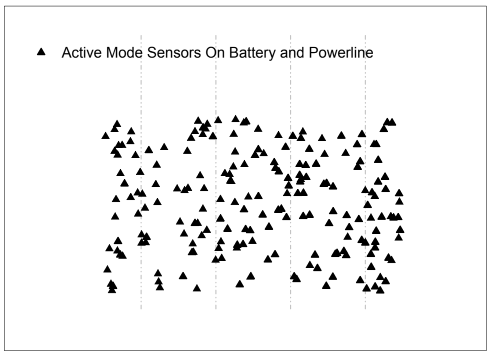
(a) Pre-defined Situation

(b) t1

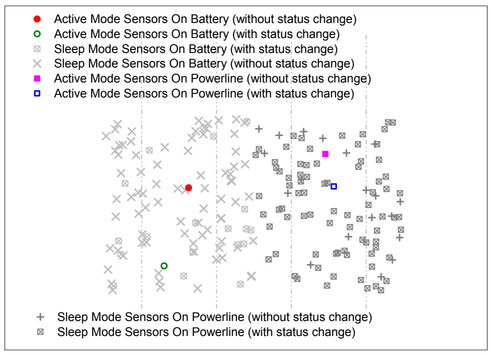
Figure 4.5: Selection of Active Sensors: System1



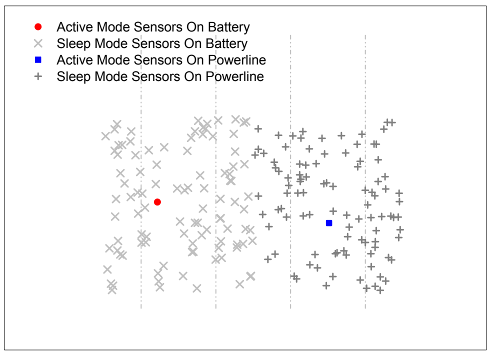
(a) Pre-defined Situation



(b) t1



(c) t2



(d) t3

Figure 4.6: Selection of Active Sensors: System2

4.4.4.2 Compare the energy saving with accuracy

From the tables above, we can see that in complex situations, the previous system is no longer useful. Thus we further compare the two versions of system under simple case when both systems are useful, case1, in order to achieve the comparison on the energy consumption and accuracy.

Fig.4.5 shows that the error rate differs a lot under different energy consumptions. When energy consumption decreases, the error rate increases obviously in the previous system, but the error rate keeps stable and low. In this case, the new version of system performs better when it requires more energy saving (less energy consumption).

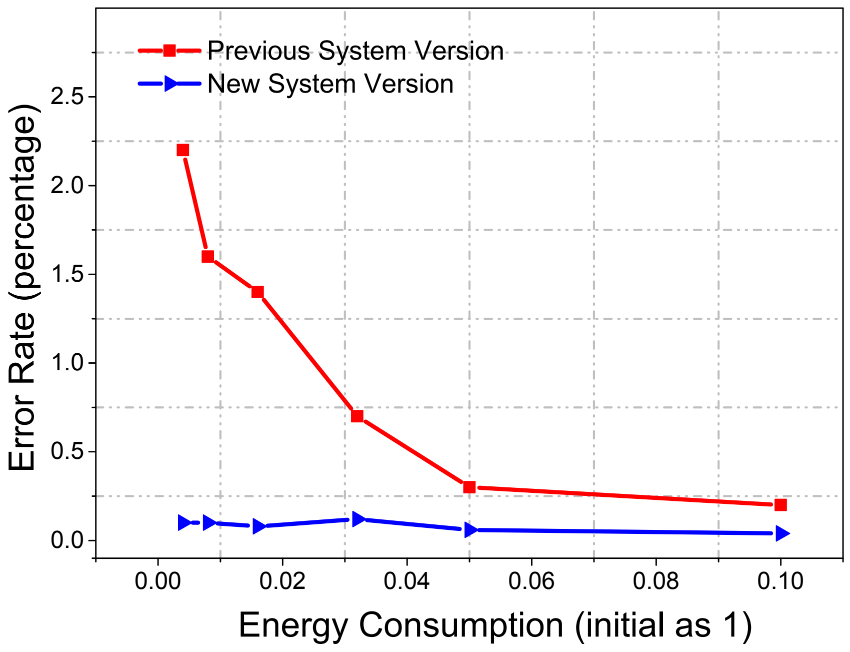


Figure 4.7: Comparison of energy consumption with error rate

4.5 Conclusion

Here, we presented an improved version of energy-efficient QoS-aware online stream data processing on Internet of Things systems. With Edge tier, our system can dynamically control the energy consumption to meet the accuracy requirement of different applications, and make this a lot better than the previous version. The simulation results show that our proposed framework is promising, and the expected improvement is achieved. In the future, we plan to evaluate our framework on real sensor nodes. Another possible improvement is to evaluate it in a scenario where the edge nodes can collaborate among themselves. Finally, we intend to come up with a better active sensor nodes selection mechanism, that besides clustering, it learns about the relationship between clusters, for example the weight for each cluster compared to the whole. Hence, we will be able to keep the situation with reduced sensors in a same combination proportion as before.

Algorithm 5 Cluster Based Active Sensors

```
Input: $x_i$
Output: $activeSensorsID$
 1: $clusters \leftarrow OnlineKmeans(x_i)$
 2: for each cluster do
 3:     $Gammas \leftarrow UpdateGamma(x_i)$
 4:     $KandS \leftarrow Combination(Gammas)$
 5: end for
 6:
 7: for Any two clusters p and q do
 8:     if $|K_p - K_q| \leq Q_K \&\& |S_p - S_q| \leq Q_S$ then
 9:         $Mergeclusterpandclusterq$
10:     end if
11: end for
12: $Updateclusters$
13: $activeSensorsID \leftarrow randomselectedfromeachcluster$
14:
15: for each cluster do
16:     $Gammas \leftarrow UpdateGamma(x_i)$
17:     $K\&S \leftarrow Combination(Gammas)$
18: end for
19: if $Admin(K, S)$ then
20:     $Turnonallsensors$
21:     $Restart$
22: end if
23:
24: function ONLINEKMEANS
25:     $RandomlygeneratedC_1, C_2, \cdots, C_n;$
26:     $ndenotethenumberofinitialcentres$
27:     loop
28:         for $eachC_p, p \in [1, n]denotethecentreid$ do
29:             $Calculate|x_i - C - p|$
30:             $Findtheminrvaluefor|x_i - C_p|$
31:             $Sensoribelongtoclusterp$
32:             $UpdateC_p : C_p \leftarrow C_p + \frac{1}{\alpha}|x_i - C_p|$
33:
34:             if $nosensorchangesthecluster$ then
35:                 return $clusters$
36:                 $Break$
37:             end if
38:         end for
39:     end loop
40: end function
```

Chapter 5

Conclusions and Future Directions

In all, we have proposed two versions of energy-efficient QoS-aware online stream data processing frameworks on Internet of Things systems.

5.1 Conclusion of the Contributions

5.1.1 On-line statistical data stream processing

In our work, we introduce an on-line statistical moments calculating algorithm which allows the efficient data processing under the computation and storage constraints. Besides, the calculation does not require the whole data stream, and the information aggregation among streams is done by performing a simple arithmetic addition. This enables the calculation can be done in a distributed way and the resources of the IoT systems can be fully utilized.

5.1.2 Energy efficient transmission control

With the consideration of energy consumption, we present two levels of transmission control in the system, on both the sensor sending and the number of active sensor nodes. The experiments have shown the significant reduction on the energy consumption under error rate tolerance.

In this case, both sensors and data that are unnecessary for presenting the real situation will not be considered into the transmission. Furthermore, this control of transmission is flexible for applications. Different requirements can be realized by,

1) predefining specific parameters (e.g. the maximum allowance of transmission time interval, the maximum data sending rate or the maximum number of sensors to be turned off.) and 2) predefining the outcome requirements such as the accuracy requirement, the energy consumption expectation. By doing this, our system provides flexible leverage towards applications.

5.1.3 Self-adaptive mechanism

With regard to the performance monitoring of the system, our system is able to use self adaption to better meet the application requirements. We firstly introduce the change detector into the system to monitor the data distribution. This feedback of the detector is used to decide how the system performs its control mechanism and is able to correct the system control decision when it is needed. This guarantees an adaptive control of the system based on the real situation.

Secondly, we mapped the window size control method used in TCP/IP to our system. By doing this, the system decides how to execute the adjustment, such as the adjustment speed, the adjustment curve, etc.

5.1.4 Overall contribution

To summarize, the overall contribution of this thesis is to propose two versions of system framework that take into consideration both the effective processing of generated data streams and the efficient usage of the available resources of devices, with adaptive monitoring mechanisms. Supported by the experiment results, the systems performance is satisfied.

5.2 Future Works

There are multiple possible ways to extend and improve the system:

1) All the parameters involved in the system can be precisely in collaborate with the system output. That is to say, we can learn about the parameters discovering how exactly the parameters affect the outcomes. In this case, the system will be able to provide simple interface to the user.

2) Function blocks inside the system can be further improved. For example, the incremented calculated results can be updated by removing the outdated result. Therefore, all function blocks can share the same filtering mechanisms that update the system running.

3) The scenario we used in this work is ideal. For example we did not take into consideration the complex geographical deployment of the sensors, such as: the overlapping of sensor coverage, the acceptable distance between sensors that allow them to be clustered. Thus, in the future, we intend to extend the scenario to a more realistic situation, considering several complexity and challenges in the real world. Also, we will need to perform the experiments on the real sensors.

Appendix A

Code for System1

A.1 Class: Main Class

```
import moa.classifiers.core.driftdetection.ADWIN;
import java.io.*;
import java.lang.reflect.Array;

import static java.lang.Math.sqrt;

public class SensorNode {

    public static void main(String[] args)
                                throws IOException {

        ReadFile readFile = new ReadFile();

        double t_step;
        double stepStart = 0.1;
        double stepEnd = 1;
        double stepStep = 10;
        int W_max;
        int maxOptionLow = 2;
    int maxOptionHigh = 3;
    int sensorNumber = 200;

    for (W_max = maxOptionLow; W_max < maxOptionHigh;
        W_max++) {

        for (t_step = stepStart; t_step < stepEnd;
            t_step *= stepStep) {
            System.out.println("W_max␣" + W_max);
            System.out.println("tStep␣" + t_step);
            int[] N = new int[200];
            double x;
            double[] moments;
            double[] gamma1 = new double[200];
            double[] gamma2 = new double[200];
            double[] gamma3 = new double[200];
            double[] gamma4 = new double[200];


            for (int line = 0; line < 20000; line++) {
//i: dataFileNumber, represent the sensors number
                for (int i = 0; i < sensorNumber; i++) {

                    x = readFile.setInput(i, line);
                    N[i] += 1;

                    gamma1[i] += x;
                    gamma2[i] += x * x;
                    gamma3[i] += x * x * x;
                    gamma4[i] += x * x * x * x;
                }
            }
```

```
                moments = moments(N, gamma1,
                        gamma2, gamma3, gamma4);
            }
        }
    }

        private static double[] moments ( int[] N,
        double[] gamma1, double[] gamma2,
        double[] gamma3, double[] gamma4){
            double m1, m2, m3, m4;
            double m1_2, c2, c2_sqrt , c2_2 , c3 , c4;
            double[] moments = new double[2];
            double combinedGamma1 = 0;
            double combinedGamma2 = 0;
            double combinedGamma3 = 0;
            double combinedGamma4 = 0;
            int totalN = 0;

            for (int i = 0; i < 200; i++) {
                combinedGamma1 += gamma1[i];
                combinedGamma2 += gamma2[i];
                combinedGamma3 += gamma3[i];
                combinedGamma4 += gamma4[i];
                totalN += N[i];
            }

            m1 = combinedGamma1 / totalN;
            m2 = combinedGamma2 / totalN;
            m3 = combinedGamma3 / totalN;
            m4 = combinedGamma4 / totalN;

            m1_2 = m1 * m1;
            c2 = m2 - m1_2;
```

```
          c2_sqrt = sqrt(c2);
          c2_2 = c2 * c2;
          c3 = m3 - 3 * m1 * m2 + 2 * m1_2 * m1;
          c4 = m4 - 4 * m1 * m3 + 6 * m1_2 * m2
                    - 3 * m1_2 * m1_2;

          //skewness;
          moments[0] = c3 / (c2 * c2_sqrt);
          //kurtosis;
          moments[1] = c4 / c2_2;

          return moments;
     }
}
```

A.2 Class: Energy Model

```
/**
 * Created by yzha8420 on 7/09/2016.
 */

public class EnergyModel {

    public static void main(String[] args) {
        double energy;
        int N ;
        N = 16000*2;
        energy = EnergyOnThings(N, 4000000);
        System.out.println(energy);
    }

    public static double EnergyOnThings(int N, int N_old){
    // N_old = Sum all lines;
```

```
double E_things;
double E_things_old;
double E_sensing;
double E_logging;
double E_processing;
double E_transmission;
double E_receiving;

//int variables_public = 1;
//int time = N;
//int the time period for the
//representing of time proportion;
//int time_old = time * (N_old / N);

int b = 1; // bit each data, initialize to 1;
int b_processing = 2 * b;

// data bit take into processing;
int F = 1;
// receiving frequency, maximum N;

E_sensing = b;
E_logging = (b/8) * (1 + 1);
E_processing = b_processing + b_processing;
E_transmission = b + b;
E_receiving = b;
E_things = N * E_sensing + N * E_logging
                        + N * E_processing
            + N * E_transmission
            + F * E_receiving;
E_things_old = N_old * E_sensing
                        + N_old * E_logging
```

```
                    + N_old * E_processing
                    + N_old * E_transmission;

        //System.out.println("E_things " + E_things);
        //System.out.println("E_things_old " + E_things_old);
        double energyRatio = E_things / E_things_old;
        return energyRatio;
    }
}
```

A.3 Class: Energy Model

```
import java.io.*;

/**
 * Created by yzha8420 on 25/07/2016.
 * This is the class to read data from file.
 */

public class ReadFile {

    public double setInput(int fileNumber, int lineNumber) {

        File f = null;
        String[] paths;
        Double data = 0.0;
        String fileName;

        try {

                f = new File("nameOfTheFileWithPath");
                paths = f.list();
                //for(int k=0; k<paths.length; k++){
                //System.out.println(paths[k]);
```

```
                    // }
                    //System.out.println(paths.length);
                    fileName = "filePath"
                        + paths[fileNumber];
                    BufferedReader reader =
                    new BufferedReader(
                            new InputStreamReader(
                            new FileInputStream(fileName)));
                    String line = reader.readLine();
                    int lineNum = 0;
                    while (line != null) {
                        if (lineNum++ == lineNumber) {
                            data = Double.parseDouble(line);
                            break;
                        }
                        line = reader.readLine();
                    }
                    reader.close();
                }

            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return data;
        }
    }
```

Appendix B

Code of System Version2

```
import moa.classifiers.core.driftdetection.ADWIN;
import java.io.*;
import static java.lang.Math.sqrt;

/**
 * Created by yzha8420 on 23/11/2016.
 */

public class Kmeans {
public static void main(String[] args) {
  double p = 0.1; //parameter for updating centers
  int n = 20000; //number of input each sensor
  int m = 200; //number of sensors
  int c = 4; //number of centers
  int[] clusterId = new int[m];
  int[] oldClusterId = new int[m];
  int[] clusterId2 = new int[m];
  int[] oldClusterId2 = new int[m];
  double x;
  double[] centre = new double[c];
  double[] centreReCluster = new double[c];
  double[] distances;
  boolean move = true;
  boolean moveReCluster = true;
  double[] kurtosis = new double[c*2];
  double[] skewness = new double[c*2];
  double[] gamma1 = new double[c*2];
  double[] gamma2 = new double[c*2];
  double[] gamma3 = new double[c*2];
  double[] gamma4 = new double[c*2];
  int[] N = new int[c*2];
  boolean reCluster = false;
  int reClusterID = 100;
  int i;

//initialize the centers for ten
  for (i =0; i<c; i++) {
        centre[i] = Math.random();
        centreReCluster[i] = Math.random();
  }

//initialize the clusters

  for (i = 0; i < n; i++) {
        int countMove = 0;
        if (i < 100 || move) {
        //loop input for each sensor
      for (int j = 0; j < m; j++) {
                x = setInput(i, j);
                distances = getDistance(centre, x);
                //get cluster for this input;
        clusterId[j] = getMin(distances);
                //get cluster average;
        centre[clusterId[j]] += p * (x -
                centre[clusterId[j]]);
                if (clusterId[j] != oldClusterId[j])
          countMove++;
                  oldClusterId[j] = clusterId[j];
          }
        //System.out.println(countMove);
        if (countMove == 0) {
          move = false;
        }
  } else {
        System.out.println(i);
        for (int j = 0; j < m; j++) {
      System.out.print(clusterId[j] + "␣");
        }
        break;
        }
  }

  for (int i2 = i; i2 < n; i2 ++){
        int countMove =0;
        if (reCluster == false) {
          ADWIN adwin = new ADWIN(.01);
          for (int j = 0; j < m; j++) {
                x = setInput(i, j);
                N[clusterId[j]]++;
                gamma1[clusterId[j]] += x;
                gamma2[clusterId[j]] += x * x;
                gamma3[clusterId[j]] += x * x * x;
                gamma4[clusterId[j]] += x * x * x * x;
                skewness[clusterId[j]] =
        moments(N[clusterId[j]], gamma1[clusterId[j]],
        gamma2[clusterId[j]], gamma3[clusterId[j]],
        gamma4[clusterId[j]])[0];
```

```
                kurtosis[clusterId[j]] =
moments(N[clusterId[j]],
gamma1[clusterId[j]],
gamma2[clusterId[j]],
gamma3[clusterId[j]],
gamma4[clusterId[j]])[1];
                if (adwin.setInput(kurtosis[clusterId[j]])
|| adwin.setInput(skewness[clusterId[j]])) {
                    if (i2> 200) {
                            reClusterID = clusterId[j];
                            reCluster = true;
                            System.out.println(i2);
                    }
                }
    }
}else {
    if (moveReCluster){
                for (int j = 0; j < m; j++) {
                    x = setInput(i, j);
                    if (clusterId[j] == reClusterID) {
                            distances =
        getDistance(centreReCluster, x);
                            //get cluster for this input;
        clusterId2[j] = getMin(distances);
                            centreReCluster[clusterId2[j]] +=
        //get cluster average;
        p * (x - centreReCluster[clusterId2[j]]);
                            if (clusterId2[j] != oldClusterId2[j])
                                countMove++;
                                oldClusterId2[j] = clusterId2[j];
                                clusterId[j] = clusterId2[j]+4;

                    } else {
                                N[clusterId[j]]++;
                                gamma1[clusterId[j]] += x;
                                gamma2[clusterId[j]] += x * x;
                                gamma3[clusterId[j]] += x * x * x;
                                gamma4[clusterId[j]] += x * x * x * x;
                                skewness[clusterId[j]] =
                        moments(N[clusterId[j]], gamma1[clusterId[j]],
                        gamma2[clusterId[j]], gamma3[clusterId[j]],
                        gamma4[clusterId[j]])[0];
                                kurtosis[clusterId[j]] =
                        moments(N[clusterId[j]], gamma1[clusterId[j]],
                        gamma2[clusterId[j]], gamma3[clusterId[j]],
                        gamma4[clusterId[j]])[1];
                            }
                        }
                        if (countMove == 0) {
                    moveReCluster = false;
                        }
                }else {
                        for (int j = 0; j < m; j++) {
                    x = setInput(i, j);
                    N[clusterId[j]]++;
                    gamma1[clusterId[j]] += x;
                    gamma2[clusterId[j]] += x * x;
                    gamma3[clusterId[j]] += x * x * x;
                        gamma4[clusterId[j]] += x * x * x * x;
                    skewness[clusterId[j]] =
             moments(N[clusterId[j]], gamma1[clusterId[j]],
             gamma2[clusterId[j]], gamma3[clusterId[j]],
             gamma4[clusterId[j]])[0];
                    kurtosis[clusterId[j]] =
             moments(N[clusterId[j]], gamma1[clusterId[j]],
             gamma2[clusterId[j]], gamma3[clusterId[j]],
             gamma4[clusterId[j]])[1];
                        }
                        clusterId =
                mergeCluster(clusterId, c, kurtosis, skewness);
                        oldClusterId = clusterId;
                        int countMerge = 0;
                        for (int j=0; j<m; j++){
                    if (clusterId[j] != oldClusterId[j])
                            countMerge ++;
                        }
                        if (countMerge != 0) {
                    System.out.println(i2 + "␣"+"merged");
                            }
                }
             }
                }
                for (int j=0; j<m; j++) {
             System.out.print(clusterId[j] + "␣");
                }
              }

            public static int getMin(double[] distances) {
              double minDistance = distances[0];
              int minLabel = 0;
              for (int i=1; i<distances.length; i++){
                if (distances[i] < minDistance) {
             minDistance = distances[i];
             minLabel = i;
                }
              }
              return minLabel;
            }
```

```
public static double[]
getDistance(double[] centre, double x) {
  double[] distances = new double[centre.length];
  for (int i=0; i<centre.length; i++) {
    double d = x − centre[i];
        distances[i] = Math.sqrt(d ∗ d);
  }
  return distances;
}

public static double
setInput(int lineNumber, int fileNumber) {
  Double data = 0.0;
  File f = null;
  String[] paths;
  String fileName;
  try {
        File("fileDirection");
        fileName =
    "fileName" + fileNumber;
        BufferedReader reader =
    new BufferedReader(new InputStreamReader(new
    FileInputStream(fileName)));
        //System.out.println(fileName);
        String line = reader.readLine();
        int lineNum = 0;
        while (line != null) {
          if (lineNum++ == lineNumber) {
            data = Double.parseDouble(line);
                break;
          }
          line = reader.readLine();
        }
```

```
            reader.close();
    } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            }
    return data;
}

public static int[]
mergeCluster(int[] clusterID,
int numberOfCluster,
double[] kurtosis,
double[] skewness)
{
    for (int i=0; i<numberOfCluster; i++){
            for (int j=0; j<numberOfCluster; j++){
                int differenceK =
        Math.abs((int)Math.floor(kurtosis[i])
        - (int)Math.floor(kurtosis[j]));
                int differenceS =
        Math.abs((int)Math.floor(skewness[i])
        - (int)Math.floor(skewness[j]));
                if (differenceK < 0.5 & differenceS < 0.5)
                        clusterID[j] = clusterID[i];
            }
    }
    return clusterID;
}

public static double[]
moments(int N,
double gamma1,
double gamma2,
double gamma3,
double gamma4) {
    double m1, m2, m3, m4;
    double $m1_2, c2, c2_sqrt, c2_2, c3, c4$;
    double[] moments = new double[2];
    m1 = gamma1 / N;
    m2 = gamma2 / N;
    m3 = gamma3 / N;
    m4 = gamma4 / N;
    m1_2 = m1 * m1;
    c2 = m2 - m1_2;
    c2_sqrt = sqrt(c2);
    c2_2 = c2 * c2;
    c3 = m3 - 3 * m1 * m2 + 2 * m1_2 * m1;
    c4 = m4 - 4 * m1 * m3 + 6 * m1_2 * m2 - 3 * m1_2 * m1_2;
    moments[0] = c3 / (c2 * c2_sqrt); //skewness;
    moments[1] = c4 / c2_2; //kurtosis;
    return moments;
            }
}
```

Bibliography

[1] Andrew Whitmore, Anurag Agarwal, and Li Da Xu, "The Internet of Things-A survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[2] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2002, pp. 1–16.

[4] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] Lukasz Golab and M. Tamer Özsu, "Issues in Data Stream Management," *SIGMOD Rec.*, vol. 32, no. 2, pp. 5–14, June 2003.

[6] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik, "The 8 Requirements of Real-time Stream Processing," *SIGMOD Rec.*, vol. 34, no. 4, pp. 42–47, Dec. 2005.

[7] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 44, 2014.

[8] Albert Bifet and Ricard Gavalda, “Learning from time-changing data with adaptive windowing,” in *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 2007, pp. 443–448.

[9] Russel Pears, Sripirakas Sakthithasan, and Yun Sing Koh, “Detecting concept change in dynamic data streams,” *Machine Learning*, vol. 97, no. 3, pp. 259–293, 2014.

[10] Gabriel Aquino, Luci Pirmez, Claudio M de Farias, Flávia C Delicato, and Paulo F Pires, “Hephaestus: A multisensor data fusion algorithm for multiple applications on wireless sensor networks,” in *Information Fusion (FUSION), 2016 19th International Conference on*. IEEE, 2016, pp. 59–66.

[11] Sangtae Ha, Injong Rhee, and Lisong Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[12] Buğra Gedik, Scott Schneider, Martin Hirzel, and Kun-Lung Wu, “Elastic scaling for data stream processing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1447–1463, 2014.

[13] Sefki Kolozali, Maria Bermudez-Edo, Daniel Puschmann, Frieder Ganz, and Payam Barnaghi, “A knowledge-based approach for real-time iot data stream annotation and processing,” in *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE*. IEEE, 2014, pp. 215–222.

[14] Melike Erol-Kantarci and Hussein T Mouftah, “Wireless multimedia sensor and actor networks for the next generation power grid,” .

[15] Hongju Cheng, Zhihuang Su, Daqiang Zhang, Jaime Lloret, and Zhiyong Yu, “Energy-efficient node selection algorithms with correlation optimization in wireless sensor networks,” *International Journal of Distributed Sensor Networks*, vol. 2014, 2014.

[16] Shan Lin, Jingbin Zhang, Gang Zhou, Lin Gu, John A Stankovic, and Tian He, “ATPC: adaptive transmission power control for wireless sensor networks,” in

Proceedings of the 4th international conference on Embedded networked sensor systems. ACM, 2006, pp. 223–236.

[17] Xiaoyu Chu and Harish Sethu, "Cooperative topology control with adaptation for improved lifetime in wireless ad hoc networks," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 262–270.

[18] Tifenn Rault, Abdelmadjid Bouabdallah, and Yacine Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Computer Networks*, vol. 67, pp. 104–122, 2014.

[19] Zhixian Yan, Vigneshwaran Subbaraju, Dipanjan Chakraborty, Archan Misra, and Karl Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *Wearable Computers (ISWC), 2012 16th International Symposium on*. Ieee, 2012, pp. 17–24.

[20] Shuai Wang, Athanasios Vasilakos, Hongbo Jiang, Xiaoqiang Ma, Wenyu Liu, Kai Peng, Bo Liu, and Yan Dong, "Energy efficient broadcasting using network coding aware protocol in wireless ad hoc network," in *Communications (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–5.

[21] Naoto Kimura and Shahram Latifi, "A survey on data compression in wireless sensor networks," in *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*. IEEE, 2005, vol. 2, pp. 8–13.

[22] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad hoc networks*, vol. 7, no. 3, pp. 537–568, 2009.

[23] Sudip Misra, Manikonda Pavan Kumar, and Mohammad S Obaidat, "Connectivity preserving localized coverage algorithm for area monitoring using wireless sensor networks," *Computer Communications*, vol. 34, no. 12, pp. 1484–1496, 2011.

[24] Efe Karasabun, Ibrahim Korpeoglu, and Cevdet Aykanat, "Active node determination for correlated data gathering in wireless sensor networks," *Computer Networks*, vol. 57, no. 5, pp. 1124–1138, 2013.

[25] Flávia Delicato, Fábio Protti, Luci Pirmez, and José Ferreira de Rezende, "An efficient heuristic for selecting active nodes in wireless sensor networks," *Computer Networks*, vol. 50, no. 18, pp. 3701–3720, 2006.

[26] Jesús MT Portocarrero, Flávia C Delicato, Paulo F Pires, Taniro C Rodrigues, and Thais V Batista, "SAMSON: self-adaptive middleware for wireless sensor networks," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016, pp. 1315–1322.

[27] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere, "Edge-centric Computing: Vision and Challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sept. 2015.

[28] Myoungkeun Choi and Bert Sweetman, "Efficient Calculation of Statistical Moments for Structural Health Monitoring," *Structural Health Monitoring*, vol. 9, no. 1, pp. 13–24, 2010.

[29] Wei Li, Flavia C. Delicato, and Albert Y. Zomaya, "Adaptive Energy-efficient Scheduling for Hierarchical Wireless Sensor Networks," *ACM Trans. Sen. Netw.*, vol. 9, no. 3, pp. 33:1–33:34, June 2013.

[30] Philippe Pébay, "Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments," *Sandia Report SAND2008-6212, Sandia National Laboratories*, vol. 94, 2008.

[31] Alexander Mcfarlane Mood, *Introduction to the theory of statistics.*, McGraw-Hill, New York, NY, US, 1950.

[32] M. Schlapfer and P. Mancarella, "Probabilistic Modeling and Simulation of Transmission Line Temperatures Under Fluctuating Power Flows," *IEEE Transactions on Power Delivery*, vol. 26, no. 4, pp. 2235–2243, Oct 2011.

[33] Malka N Halgamuge, Moshe Zukerman, Kotagiri Ramamohanarao, and Hai Le Vu, "An estimation of sensor energy consumption," *Progress In Electromagnetics Research B*, 2009.

[34] Angie King, "Online k-means clustering of nonstationary data," *Prediction Project Report*, 2012.

[35] Mingzhou Song and Hongbin Wang, "Detecting Low Complexity Clusters by Skewness and Kurtosis in Data Stream Clustering.," in *ISAIM*, 2006.