SAMSUNG
Best Paper Award 2017

Design and Implementation of A Distributed, Fault-tolerance and Scalable IoT Edge Computing Platform

Binger Chen[1]

(Abstract)Internet of things (IoT) has been widely used in plenty of areas. We are confronted with an era of IoT data explosion. Abundant amounts of data processing requests are sent to cloud, which suffers from high storage and traffic costs. To alleviate pressure of cloud and network bandwidth, reduce the latency, protect user's privacy and improve the capacity of real-time processing, edge computing has emerged. This paper design and implement a lightweight edge computing platform which is high distributed, fault-tolerance and scalable. We propose a novel task scheduling approach among edge devices to ensure the robustness of execution. The experiment demonstrates our platform is high-available.

1. INTRODUCTION

In recent years, IoT has been developing dramatically in every industrial area, as we have seen a huge increase in smart home, smart building and smart city[1]. Huge amounts of IoT devices are installed, which have created large date volumes. These data can request massive services, only a few of which are executed locally, most rely on cloud's ability. Devices will transmit data to cloud for analysis and retrieve results from cloud. Cloud Computing and IoT have been converging ever stronger[2]. The characteristics of cloud computing such as large-scale, high reliability, versatility, scalability and easy to maintain provide these substantial IoT devices solid computing infrastructure.

However, cloud computing has some inevitable problems[3][4][5][6]. First of all, the amount of data produced by substantial devices to be carried to the cloud can be significant, which will cause transmission traffic and cloud storage pressure. Second, intensive computing workload is a severe test on cloud's capacity. Third, cloud computing will cause latency on some tasks that need to be executed in real-time, such as handling emergency and instant information. Last but not least, moving data away from end users might severely interfere user's privacy. For all these reasons, more and more concepts are brought forward to push computing to the edge of networks, such as fog computing proposed by Cisco, as the name suggests is offloading cloud's workload on small computing centers in local area. Furthermore, people consider pushing computing closer to things and data sources, which lead to edge computing. Enhancing capacity of cloud and transmission technology and other hardware infrastructures can indeed solve the problems described above. But it's a challenging mission and most crucial circumstance is, substantial idle devices and computing resources are wasted. Effectively leveraging these resources is another significant motivation to study edge computing.

In current studies on edge computing, most of them focus on pushing workload to edge computing centers, which usually are stationary device, such as edge base station, small routers and other IoT hubs[7][8][9][10][11][12]. But in fact most edge devices are with the feature of mobility. In this condition, these works still address fog computing essentially, without pushing computing to real edge of network. Although there have been significant effort on offloading tasks on mobile devices, Mustafa Y A et al. migrate tasks on charging smartphones[13], Hoque M A et al. ensure sensor data are processed completely at the end devices[14], they either restrict the mobility of devices, or implement simple tasks like data storage and query, or unable to schedule tasks on edge devices flexibly. This paper address many of these challenges to develop a flexible scheduling, high distributed and scalable lightweight computing platform.

The platform we design is not a traditional computing center, but a tasks and devices management center. The main functions of the platform are as follows:

- Serve as local network communication hub;
- Manage accessed edge devices;
- Complete service register of devices, requesting tasks from cloud;
- Schedule and assigning tasks flexibly based on devices' status information;
- Cache small-scale data, compress and reduce abundant data sent to cloud;
- Allocate priority to tasks needed to be executed in real-time;
- Control devices based on data processing results;
- Equipped with high fault-tolerance mechanism to dispose the instability of mobile edge devices.

Specifically, we set the IoT hub as a Wi-Fi router which can connect every local device to Internet[15]. A device can register at the hub when requesting a certain service. Then hub requests and retrieves the tasks needed to be executed from the cloud to accomplish the service. Given the restrictions of edge devices, we design a lightweight job scheduler instead of employing existing heavy framework such as Apache Storm etc. There're two main types of services.

First, when the service involves data analysis, hub will collect and maintain every device's status information (i.e. usage of CPU, RAM, I/O, energy etc.) in real-time, then predict the task execution times in zero load conditions to calculate the current computation resources. We propose a novel load balancing algorithm to flexibly scheduling multiple tasks which are currently requested. In particular, we place emphasis on fault-tolerance mechanism due to the instability of edge devices. They may leave or join in current network at any moment or be executing heavy local tasks by their owners. To maintain the robustness of execution and normal use experience of edge devices, we design a fault-tolerance framework which can auto-suspend tasks, rapidly cache and migrate task progress.

[1]Advanced 2 Lab, SRC-Nanjing, Building 4, Chuqiao City, Andemen Street No.57, Nanjing, 210019, China

Second, when the service involves transmitting data to cloud, we design a data reduction and compression module to filter vital data.

The main contributions in this paper contains:

1. Design and implement a high scalable lightweight edge computing platform;
2. Propose a novel distributed task scheduling and fault-tolerance mechanism;
3. Demonstrate the viability and efficacy of the platform in multiple experiments.

The rest of this paper is organized as follows. Section 2 lists the challenges we dedicate to overcoming during designing and implementing the platform. Section 3 overviews the logic of the architecture, and describes the technical details. The platform's prototype implementation and basic performance evaluation are provided in Section 4. We conduct two experiments to test the platform. In Section 5, we conclude the shortage of our solution and propose the future work.

2. CHALLENGES

The platform we intend to design needs to fulfill data transceiver, data processing, data storage, task scheduling, network communications and a series of functional modules, so a rigorous logic of architecture is required in order to make each module seamlessly connected and adapting to the whole system.

The platform is installed on the central node of an edge network. Considering the limitations and instability of edge infrastructure, the platform must be as lightweight and scalable as possible. Therefore we cannot directly employ the existing mature open source distributed streaming processing framework. We have to seek the simplest solution to realize these complicated features and functions.

Moreover, how to schedule tasks among multiple mobile devices, and how to adapt assigned tasks to different kinds of devices is another concern of our study.

Since the edge devices are not dedicated to executing IoT data processing tasks, the status of these devices is quite unstable. Given the fact of this circumstance, to ensure the robustness of the system, we should not only focus on leveraging the 'idle' computing resources, but also take into account the fault-tolerance mechanism when assigning tasks.

Because of the limitation of resources in edge network, it cannot store large amount of data. And mass storage is inconsistent with the purpose of edge computing. Therefore, real-time processing of intensive data stream is a formidable challenge. The effect of data processing should also be guaranteed.

3. IOT EDGE COMPUTING PLATFORM

3.1 System Overview

IoT edge computing platform is a streaming processing system deployed at the edge of internet, which can handle IoT streaming data near their sources and flexibly schedule data processing tasks to nearby idle computing resources in the network. The system logic is shown in Figure 1. The dotted line is the process that only be performed when device first accessing to the network, and the solid line is the regular workflow. Edge devices refer to devices offering idle computing resources.

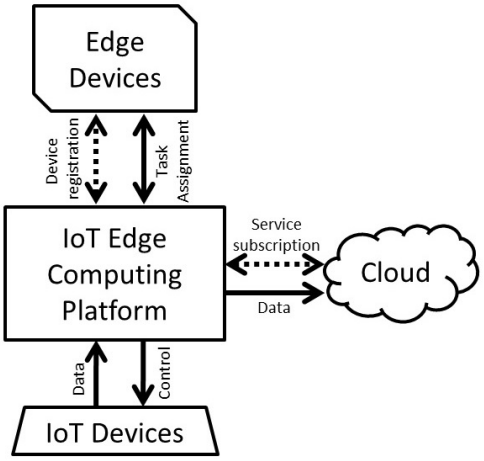

Figure 1. System Overview

User can subscribe the services they need from cloud when configuring the system for the first time. The cloud will send the program of tasks supposed to be executed in the service, as well as the rules of trigger engine (if response to the results of processing is required) to the edge computing platform. Due to our system is placed in the IoT environment, we utilize IoT hub to carry our platform. The platform stores these tasks as well as the types of IoT devices which producing the input data of these tasks. Once the data from a certain type of device is received, the corresponding processing program will be executed.

In addition, since our intention is to leverage idle computing resources to perform the required tasks rather than limiting the program on a central node, we need to consider the communication between edge devices and our platform. In general, edge devices are a kind of mobile device which is common and well-equipped in the IoT environment (a most typical instance is the smart phone). The IoT hub is also a local network hub, via which the devices can be connected to the Internet. When a edge device is connected to the IoT hub for the first time, the edge computing platform will add it to a device list library. The device list library can dynamically store all of the edge devices currently available in the network, as well as their status such as load capacity. Meanwhile, the status monitoring module is delegated to these devices by the platform. This module is used to monitor the device status and report it to the platform according to user's requirements. The status information will then be updated in the device list library.

The platform's regular workflow is as follows: If user has subscribed the service of processing data from a certain IoT device such as a sensor, when the system receives streaming data transmitted by this device, it will extract the program of tasks retrieved from the cloud and dynamically schedule these tasks to the edge device according to their current status. These devices will send the processing results back to the system when completing the execution. If there's a trigger engine rule to the IoT device, the system will take the corresponding response action, otherwise the results will be directly uploaded to the cloud.

In our work, we also equip the edge computing platform with data reduction module. For those data which is not needed to be processed or cannot be processed on the edge device, the system will perform real-time streaming reduction on them according to the configuration, then uploaded the compressed

data to the cloud. In this manner, we can achieve the goals of reducing the transmission bandwidth and relieving cloud storage pressure. The data filtering algorithm is also pre-fetched from the cloud by subscription.

3.2 Target Application

According to the above statement, the platform is suitable for the scenarios of real-time data processing and cloud's pressure offloading. There are several application instances of IoT edge computing platform:

- Anomaly detection of machine status in smart factory. In smart factory, masses of sensors are installed to monitor the status of machines for anomalies detection. These sensor data are analyzed to obtain the machine's current situation. For anomalies, the normal case is to detect and deal with them as soon as possible. The best case is real-time monitoring. If transmitting these large amounts of sensor data to the cloud, then processing them and returning the results to responding terminal, the network latency will lead to a lot of emergency cannot be dealt with in time. Our platform is able to solve this problem by directly processing data near the sensors, which can significantly reducing the latency caused by communicating with cloud. The same scenario can also be applied in smart building, smart city and so on.

- Real-time context-aware appliance automation in smart building. Internet of things is invented for bringing convenience to human's daily life, i.e. automatically regulating the living environment to reduce the frequency of manually setting. In smart building the appliances in a certain region can be automatically set based on human activities. For instance, we can process the pictures from surveillance camera to analyze current activity which can lead to the auto-control of corresponding appliance. The simplest scenario is turning on the lights when working and turning off the lights when resting. We know that the size of picture and video files is quite large. If they are uploaded to the cloud for analysis, it is likely to cause significant latency and the response cannot be made in time. Moreover, uploading the surveillance videos to the cloud might cause the issue of privacy violations. Under this circumstance, edge computing platform is needed to solve these problems locally.

- High frequency streaming data reduction. There are many cases where the data does not need to be processed in real-time, or for the time being only need to be stored for further processing. However, many IoT devices are sampled at high frequencies, such as one data point per millisecond. These large volumes of data may cause uploading traffic. It is obviously that not every millisecond of data is useful. There will be a lot of redundant data stored in cloud, which brings transmission and storage costs. If implementing data reduction mechanism in the edge computing platform, this problem will be solved by decreasing the amount of data to be uploaded and the transmission bandwidth will be saved.

3.3 Architecture

Figure 2 is the system architecture of IoT edge computing platform. The dashed box contains the modules in the platform. On the one hand, the IoT devices generate data and upload the data with their device ID to the platform. Then the platform will retrieve the corresponding processing tasks according to the services subscribed by user. On the other hand, the edge devices which can provide idle computing resources and are currently available will update their device ID and status in device list library. The task scheduling module dispatches the computing tasks and distributes them to edge devices according to the features of tasks and the status of computing resources. These edge devices will upload the processing results to the platform when completing the execution of the task and the platform then transfers them to the cloud. If the results are not needed to be upload and should be responded locally, the platform will perform corresponding control on IoT devices in real-time based on pre-configured trigger engine rules. The control module is beyond the scope of this paper.

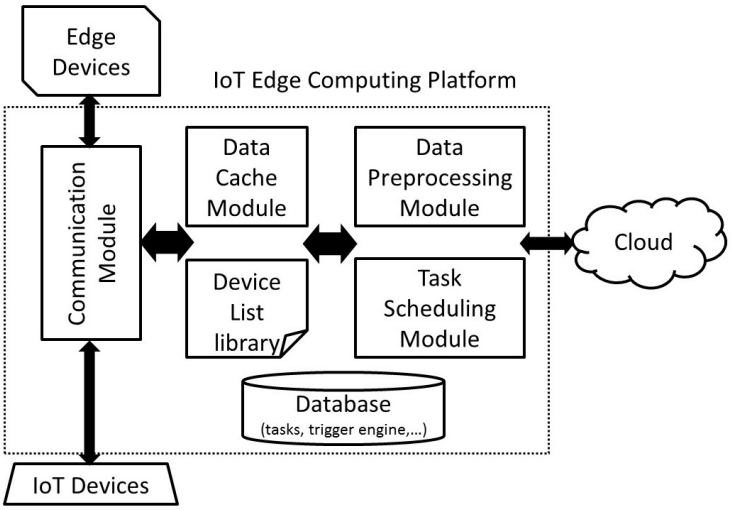

Figure 2. Architecture of IoT edge computing platform

If the uploaded data needs to be preprocessed before it can be used as input to the program of tasks, the data will be normalized first in the preprocessing module. The preprocessing algorithm is also retrieved from the cloud when first time configuring. In addition, the data reduction process is also implemented in this module. If no further processing is required, the compressed data will be directly uploaded to the cloud. The data reduction rules are preset based on the type of the device which is producing data.

If the data stream is transmitted in high frequency or the amount of data is enormous, and the available computing resources currently in the network cannot afford to perform all the data processing tasks, the data will be queuing cached in the data cache module based on priority and chronological order. The data priority is preset by user and stored together with tasks in the database. In order to build lightweight and scalable data cache module and task database, we utilize SQLite to store data. This paper does not focus on discussing data storage.

The data cache module is also used for fault-tolerance mechanism to cache the latest processing results, the newest data model, and the data that is being processing, so that the task can be dispatched to other edge device in a timely manner when current device is suddenly unavailable. And due to the latest results and progress are cached, there is no need to restart tasks from scratch on new device. The fault-tolerance mechanism will be described in the following context.

Figure 3 shows the platform's modules implemented on edge devices. These modules are transferred to the devices when they first time access to the platform. First is the device status monitoring module, which can obtain the CPU, RAM, I/O and energy capacity in real-time, then calculate the current load. This information will be uploaded to central platform through the communication module. According to this received knowledge, the central node schedules tasks and dispatches them to the data processing module on edge devices, then waits to receive the processing results. In addition, the communication module is also responsible for transmitting heartbeat packets to the central node to confirm the survivability for fault-tolerance mechanism.

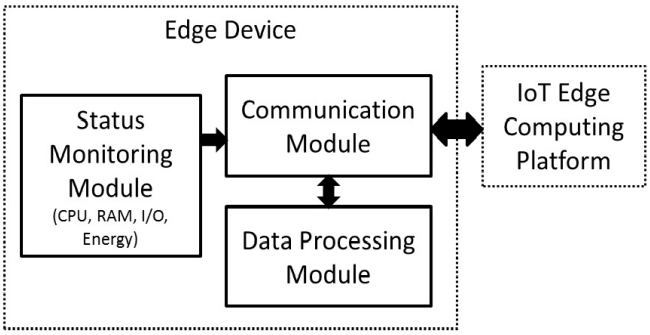

Figure 3. Modules on edge devices

The technical details of our platform are discussed below:

3.3.1 Communication Module

Since our platform is installed on the IoT hub in edge network, a lightweight system design is required. To achieve this goal, we choose the MQTT protocol in the communication module[16]. It is a lightweight low-bandwidth transmission protocol that satisfies the design logic of edge computing platform. This solution can not only save the transmission bandwidth, but also break the resource limitations on edge nodes. Moreover, MQTT also provide publish/subscribe transport mechanism which is in line with our platform architecture described above. MQTT can also handle heterogeneous features of devices in IoT environments. And it is a highly scalable and flexible framework that can be implemented in different systems. We leverage Mosquitto to fulfill the transmission and distribution of streaming data through MQTT in IoT edge network.

3.3.2 Task Execution on Edge Devices

Our platform is dedicated to distributing computing tasks to the edge devices accessing to the network. These devices which can offer idle computing resources are usually smart phone, tablet, laptop, smart TV and so on. They are quite common in IoT environment and capable of high-performance computing. They also can be connected to the Internet. Last but not least, they are not entirely occupied most of the time. There are two conditions that need to be considered when scheduling tasks among these devices:

- Tasks can be automatically executed in the background. Due to the tasks assigned by the platform are irrelevant to the device owner's daily use, it can neither be manually started by device owner, nor affect the owner's normal use. Thus these tasks must run in the background.
- The program of tasks should be able to run on multiple platforms. Since the type of edge device is not fixed and can change at any time, the program of task sent to the edge device must be platform-free and capable of directly running.

Given these two conditions, we choose Java as the programming language of task's executable file. It is a cross-platform programming language. This paper only consider Android and Android kernel platform. Furthermore, we take advantage of the Java reflection mechanism; it can get all the attributes and methods of an arbitrary Java class, and can call any attributes and methods of an arbitrary Java object. The core of its function is that the object's method is dynamically called, so that the Java class can be instantiated at runtime.

We use the Java reflection API for android to install reflection mechanism as Android background service on edge device. By this means, no matter what kind of tasks is assigned to the device, it can automatically load and execute.

As for the executable file sent to the edge device, we utilize the latest released Google Android tool chain Jack & Jill[17][18]. Jack is an Android Java compiler tool, Jill is Jack's middle library linker, they can directly compile Java files (.java) and third-party library files (.class) to .dex format files, which can be packaged into .jar then sent to the edge device. Through the reflection mechanism the program can directly run on the device. This is a proper solution to the two issues proposed above.

3.3.3 Task Execution on Edge Devices

A. Predicting the compute capability

To perform reasonable task scheduling, first we need to know the compute capability of each edge device to each task. We use the duration time of executing the task in zero load condition to indicate the compute capability. It is related to the device's performance (Note that the zero load condition dose not involve those necessary system processes.). Here we only consider the impact of CPU frequency and RAM size. P refers to the device performance. We adopt the weighting method to get the P value:

$$P = k_1 p_{CPU} + k_2 p_{RAM} \tag{1}$$

p_{CPU}, p_{RAM} is the value of CPU frequency and RAM size. k_1, k_2 are their weights on the performance. We assume that device's compute capability to the same task is linear to its performance. Then if the performance P' of a certain device and the compute capability c'_{s_i} to a certain task s_i (i.e. reciprocal of execution time at zero load) are already given, we can calculate the compute capability to task s_i of an arbitrary device x with performance P_x:

$$c^x_{s_i} = c'_{s_i} * P_x / P' \tag{2}$$

For different types of devices we set one device as a reference sample to obtain P' and c'_{s_i}. Then we experiment with ten devices and run the same task on them. By compared the experiment data with the reference data, the weight values k_1 and k_2 are corrected. In this way, for any newly joined device, we can use equation (2) to calculate its compute capability to task s_i.

Although this method is not quite rigorous, it can satisfy

the basic practical needs with minimal experiment data in the simplest manner.

B. Task scheduling method

- Current load of the device. When calculating the device load, different metrics selected determine the accuracy of the value. In this paper we choose three metrics which are most consumable and influential: CPU, RAM and I/O. We use weighting method to calculate the load[19]. Every load metric is assigned a weight and the load of each device is:

$$L = \sum_{i=1}^{3} (k_i * X_i), (0 \leq L < 1) \quad (3)$$

  X_1, X_2, X_3 are the occupancy rate of CPU, RAM and I/O, respectively. They are obtain by the status monitoring module on edge device. k_1, k_2, k_3 are their weights which can be dynamically adjusted.

- Current compute capability of the device. Using the above method we can already predict the compute capability c_{s_i} to task s_i at zero load. Apparently when the load is L, its compute capability to task s_i is changed to:

$$C_{s_i} = c_{s_i} * (1 - L) \quad (4)$$

- Double threshold. In our method, we use double threshold by setting an upper limit L_{max} and a lower limit L_{min} for device load. If the load value is greater than L_{max}, the device status is set as 'busy'. If the load value is less than L_{min}, the device status is set as 'free'.

- Task scheduling. As referred above, we implement a device list library in our platform. It stores the information of current available devices in the network as shown in Table 1. This library contains device ID, device load, compute capability to each task, busy/free status, tasks currently executed and tasks in queue.

Table 1. Device list

Device ID	Device load	Compute capability to each task	busy /free	Tasks currently executed	Tasks in queue
Device 1	...	C_{s1} C_{s2} ...	...	...	...
...					

  The current load of the device is periodically reported to the platform at regular intervals to reduce power consumption. This time interval can be dynamically adjusted. If the state of a device changes little for a long time, the time interval increases. If the state of the device changes dramatically, such as changed from one state stage (busy/free/normal) to another, the load information will also be reported.
  When a new task is needed to be scheduled, the platform will sort the devices in list based on their compute capability to this task, together with tasks currently executed on them and tasks in queue. Then the platform selects the most appropriate device to perform the task. If the current status of a device is busy, this device won't be not considered. If there are devices with free status in the device list, only these devices are considered.

3.3.4 Fault-tolerance Mechanism

Since edge devices are mostly mobile devices and we are utilizing the idle computing resources of these devices, the devices are likely to be unavailable at any time such as leaving the network or turning to the busy state when their owner start a heavy application. In this circumstance, the platform's fault-tolerance mechanism is particularly important. Here we adopt the method of caching the latest processing results and data model.

Every time a new task or new data model are delivered to the edge device, the platform will cache them in the database until the edge device returns the processing results, then the memory space is released. If the device returns busy status or loses connection before completing the task, it will be excluded from the scheduling. The device list library will also be updated, and the task will be re-scheduled. Once the new working device is determined, the contents in cache will be migrated to it.

For those devices which are currently working, they will return the newest intermediate results and data models at some certain time nodes. This information describes the current progress of the task and is stored in the cache. In this way, once a working device is not any more available, the task doesn't need to be started from the beginning after re-scheduled.

3.3.5 Data Reduction

In order to relief the pressure of data transmission and cloud storage, we add a data reduction mechanism to the platform. It can trigger the corresponding data reduction method according to the device type, then compress the data and upload them to the cloud. We will perform an experiment of edge streaming data reduction in the next session to illustrate.

4. IMPLEMENTATION

We implement our platform on Raspberry Pi 3, employ two Android smart phone (Galaxy A9: 1.8GHz,3GB and Galaxy S7: 2.15GHz,4GB), one Android tablet (Galaxy Tab S3: 2.15GHz,4GB) as edge devices, and utilize Amazon AWS to simulate cloud[20]. Our platform is a super lightweight system which is only 16MB. We design two experiments to demonstrate our platform. First is detecting anomaly of streaming sensor data in real-time. We use an incremental learning algorithm scheduled on different edge nodes in different device status we simulate. Results show the platform can efficiently and steadily output feedback in real-time. Second is reducing streaming data to reduce transmission bandwidth. We adopt major minima and maxima reduction algorithm to filter data[21]. Results show we not only dramatically reduce data volume, but also maintain the vital data features.

4.1 Edge Anomaly Detection

We employ the data from DEBS Grand Challenge 2017. The data set comes from two types of machines in smart factory. They are equipped with sensors that measure various parameters of a production process. All the measurements are taken at a certain point in time. We input these data in a streaming manner to our platform.

We perform incremental learning on the received data based on the multi-layer perceptron model of neural network. The flow chart is shown in Figure 4. First, we initialize a multi-layer perceptron model which is learnt from the histori-

cal data. We set the length of a data window. The input of the model is the data window, and the output is the predicted value of the data adjacent to the window. The length of the window is fixed, which means it is a tumble data window. Once a new data is added in the window, the first data will be deleted.

When streaming data is transmitted, it will fill the window and be input to the neural network to predict the data value at the next time point. If the difference between the true data and the predicted value is within a preset range, the data is considered normal, and the learning model is updated by the real data which is inserted into the data window. If the difference is beyond the preset range, the data is considered to be anomalous, and the response action is triggered. Meanwhile the predicted value will be inserted into the data window to start the next round of prediction.

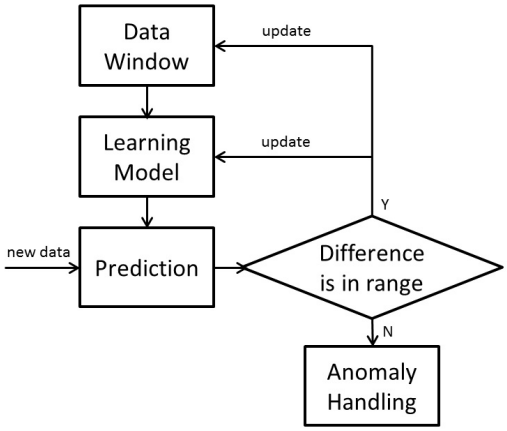

Figure 4. Workflow of anomaly detection

In the experiment, we schedule different sensor data anomaly detection tasks among the edge devices. Data transmission frequency is set to 20Hz. We assign some data reading and writing tasks to the devices in advance in order to simulate the multi-task distribution environment, and perform the experiment in the following three cases. Two metrics are calculated: 1. Anomaly detection accuracy; 2. the delay time of the process (start from transmitting data to the platform, end in receiving the prediction from the edge device). The three cases are:

- Case 1: Three edge devices are in normal state.
- Case 2: During executing of the tasks, one of the device starts to run a heavy application.
- Case 3: During executing of the tasks, one of the device loses connection.

We use 50% of the data set to initialize a multi-layer perceptron model, the length of data window is set to 100 data points. The accuracy of the model turns out to be 92%. The final results are shown in Table 2. It can be indicated that 1.the accuracy of the three cases can all meet requirements and the result in the first case is even better than the initial model; 2.In the second and third case, the fault-tolerance mechanism is well-performed except a slight delay; 3.The delay time of the three cases is within the tolerable range in practical.

Table 2. Experiment result of edge anomaly detection

Case	Avg. Accuracy	Avg. Delay time (ms)	Max. Delay time (ms)
Case 1	*93.82%*	*85.91*	*396.23*
Case 2	*89.1%*	*95.34*	*436.25*
Case 3	*90.33%*	*97.82*	*441.38*

4.2 Edge Data Reduction

We employ the major minima and maxima algorithm to perform real-time streaming data reduction. The data set is from a temperature sensor in real environment. We sampled data for two weeks. We use the method in [21], the main idea is to only upload the major maximum and minimum points. The algorithm can determine whether a data is the major maximum or minimum points in real-time and reduce the data stream. We implement this method in our platform and set the event trigger mechanism. When the platform receives a sensor data sent in simulation, it begins to perform real-time reduction. Result shows that the data volume can be compressed by 60%. We use the compressed data to fit the data curve, which is consistent with the trend of the original plot, and maintains the significant features.

5. FUTURE WORK

This paper designs a distributed, fault-tolerance and scalable IoT edge computing platform that can satisfy the proposed functional requirements in practice. However, there are still some imperfections need to be further improved.

First of all, when the task is delegated to the edge device for executing, the device still needs to install the reflection mechanism in advance. How to employ newly accessed device to directly run the task program without pre-setting is still a formidable challenge.

Secondly, we did not take into account the data transmission time and network status during the task scheduling process. In the next step, we will consider how the network status of different devices can affect their performance on edge tasks.

Furthermore, there are some other situations such as the failures of IoT hub haven't been discussed in the fault-tolerance mechanism. How to guarantee the system normal operation in this case will become a vital issue in future work.

In this study, the experiment scenarios and data are still relatively insufficient. In future we need to further evaluate the platform's robustness, scalability and other performances in multi-device and multi-task environment. We will also attempt to distribute tasks with high resource consumption and dispatch larger size data such as images and videos to test platform's capacity.

REFERENCES

[1] Zhao S. Li S, Da Xu L. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, (2015).

[2] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *Hot Topics in Web Systems and Technologies*, pages 73–78, 2015.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Edition of the Mcc Workshop on Mobile Cloud Computing*, pages 13–16, 2012.

[4] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. *Fog Computing: A Platform for Internet of Things and Analytics*. 2014.

[5] Mohammad Aazam and Eui Nam Huh. Fog computing and smart gateway based communication for cloud of things. In *International Conference on Future Internet of Things and Cloud*, pages 464–470, 2014.

[6] Salvatore J Stolfo, Malek Ben Salem, and Angelos D Keromytis. Fog computing: Mitigating insider data theft attacks in the cloud. In *IEEE Symposium on Security and Privacy Workshops*, pages 125–128, 2012.

[7] Apostolos Papageorgiou, Bin Cheng, and Erno Kovacs. Real-time data reduction at the network edge of internet-of-things systems. In *International Conference on Network and Service Management*, pages 284–291, 2015.

[8] Hooman Peiro Sajjad, Ken Danniswara, Ahmad Al-Shishtawy, and Vladimir Vlassov. Spanedge: Towards unifying stream processing over central and near-the-edge data centers. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 168–178, 2016.

[9] Jongwon Yoon, Peng Liu, and Suman Banerjee. Low-cost video transcoding at the wireless edge. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 129–141, 2016.

[10] Mostafa Uddin, Ibrahim Ben Mustafa, and Tamer Nadeem. Edgeeye: Fine grained traffic visibility at wireless network edge. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 111–112, 2016.

[11] Qingyang Zhang, Zhifeng Yu, Weisong Shi, and Hong Zhong. Demo abstract: Evaps: Edge video analysis for public safety. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 121–122, 2016.

[12] José Angel Carvajal Soto, Marc Jentsch, Davy Preuveneers, and Elisabeth Ilie-Zudor. Ceml: Mixing and moving complex event processing and machine learning to the edge of the network for iot applications. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 103–110, 2016.

[13] Mustafa Y Arslan, Indrajeet Singh, Shailendra Singh, Harsha V Madhyastha, Karthikeyan Sundaresan, and Srikanth V Krishnamurthy. Cwc: A distributed computing infrastructure using smartphones. *IEEE Transactions on Mobile Computing*, 14(8):1587–1600, 2015.

[14] Mohammad A Hoque and Sasu Tarkoma. Sensesensing data analytics at scale beyond the edge network. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 77–78, 2016.

[15] Peng Liu, Dale Willis, and Suman Banerjee. Paradrop: Enabling lightweight multi-tenancy at the networks extreme edge. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 1–13. IEEE, 2016.

[16] http://mqtt.org/.

[17] https://android.googlesource.com/toolchain/jack.

[18] https://android.googlesource.com/toolchain/jill.

[19] Min You Wu. On runtime parallel scheduling for processor load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):173–186, 1997.

[20] http://aws.amazon.com/.

[21] Eugene Fink, Kevin B Pratt, and Harith Suman Gandhi. Indexing of time series by major minima and maxima. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 3, pages 2332–2335. IEEE, 2003.