2022 Tenth International Conference on Advanced Cloud and Big Data (CBD) | 979-8-3503-0971-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/CBD58033.2022.00058

IoTPass: IoT Data Management System for Processing Time-series Data

Zehua Nie[1], Can Su[1], Yichen Mao[1], Kaigui Bian[1,2]
[1]School of Computer Science, Peking University
[2]National Engineering Laboratory for Big Data Analysis and Application
{1900017708, cansu, 1900012906, bkg}@pku.edu.cn

***Abstract*—With the increased popularity of IoT devices and applications, the amount of time-series IoT data has exploded and the need for efficient processing and analysis of time-series IoT data has also emerged. In this paper, we present, IoTPass, an IoT data management system for processing time-series data. IoTPass employs the microservice architecture to facilitate the development and deployment of functions, and adopts the Object model to structure the data and facilitate data analysis and visualization afterwards. IoTPass also adopt Revert-RPC (Remote Procedure Call) communication technique to avoid the performance degradation problem in service call processes and supports both synchronous and asynchronous service invocation modes. Experimental results demonstrate that IoTPass system meets the requirement of processing time-series IoT data in real time, which shows the effectiveness and scalability of the system.**

***Index Terms*—IoT, data management, microservice**

I. Introduction

The Internet of Things (IoT) has been widely deployed in many applications, including smart home [1], environmental monitoring equipment [2], smart agriculture system [3], to serve as a system that connects all items with the Internet through information sensing and exchanges to achieve intelligent identification and management. The concept was first mentioned by Kevin Ashton in a speech to Procter & Gamble (P&G) in 1999 [4] and has developed rapidly over the past few decades. Nowadays, there are massive real-time data being generated and sent to the cloud continuously with the rapid proliferation of smart devices. The IoT offers several benefits and opportunities to organizations including the ability to access information from anywhere at any time on any device, and reduce the need for human intervention, saving both time and money.

The core challenge for IoT data management is how to guarantee the low latency when processing data requests/responses from/to massive number of parallel connections of IoT devices in real time. Compared with traditional Internet data, IoT data has the following characteristics.

- **Time-series data**: IoT data is generated continuously by connected devices according to a set period or triggered by external events. Each row in tables of time-series database has a field named timestamp and it is particularly important for data analysis afterwards.
- **Structured data**: Traditional Internet data such as social media data are unstructured, yet IoT data are often structured and numerical. As an example, the current and voltage data collected by smart meters can be represented by 4-byte standard floating-point numbers.
- **Unique and periodic data source**: The data collected by one IoT device is completely independent of another. Besides, the collected data generally has a retention policy based on the duration, and most of users are concerned about the trend/pattern of data over a period of time rather than the information at a specific time point.
- **Massive amount of data**: The amount of data generated by the IoT devices is extremely huge. A common IoT system producing 1 million events per second will generate more events in a single day than the total number of US stock market trades in a year [5].

Traditional relational databases have been used to handle the most challenging problems of data management [6] [7]. However, tremendous volume of data in IoT workloads plays against the strengths of traditional relational databases. Hence, time-series databases have received great attentions recently, and become a better choice for processing IoT data. For example, InfluxDB [8] provides an SQL-like language without external dependencies; TimeScaleDB [9] supports full SQL and advanced time-series analytics; Apache IoTDB [10] provides an optimized columnar file format for efficient time-series data storage.

However, an ideal IoT data management system is supposed to connect upper-layer IoT applications and the time-series databases, as a middleware. Based on the above characteristics of IoT data, the IoT data management system need to meet the following design requirements: (1) an efficient distributed system that supports rapidly data processing; (2) low latency under the premise of real-time processing and strategic decision; (3) high configurability and scalability that can accommodate servers and storage devices of different configurations; and (4) comprehensive supervision and maintenance system that supports accessing new devices and risk monitoring.

In this paper, we present, *IoTPass*, an IoT data management system, which has the following advantages compared with existing time-series database systems.

- **Efficient connection between devices and cloud**: The core message processing module in IoTPass adopts a distributed architecture without a single point dependency. If the message fails to be sent, it can be automatically

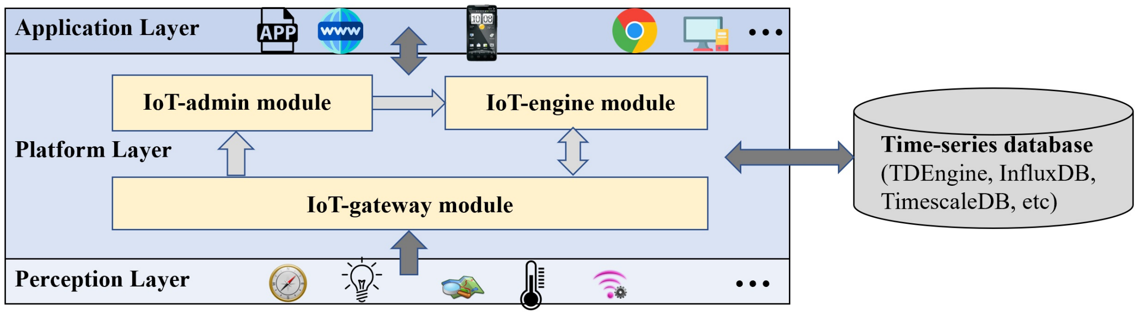

Fig. 1. The architecture of IoTPass system.

retried to ensure connection reliability.

- **A suite of data interfaces**: IoTPass system provides open interfaces which facilitates the data interaction from devices to IoTPass platform and then to the upper-layer applications.
- **Device management and real-time monitoring**: Our system supports flexible access to new device and provides a comprehensive monitoring and alert system which can detect abnormal events of devices.

The rest of the paper is organized as follows. We give the overview of our system in Section II. The methodology is presented in Section III, followed by details of implementation in Section IV. We show the experimental results in Section V and conclude this work in Section VI.

II. System Overview

The design of an IoT data manegement system can be divided into three different hierarchical layers, namely the sensing layer, the platform layer, and the application layer, as shown in Figure 1.

1) The perception layer collects data from connected devices, converts the data into a specified format, and send it to the platform later.
2) The platform layer receives, processes, relays messages and stores the data into the time-series database.
3) The application layer provides different data services to the user through a suite of data interfaces.

The architecture of IoTPass system is shown in Figure 1, the platform layer of which is mainly composed of the following four modules:

- **IoT-gateway** is responsible for unified access and scheduling of devices and messages.
- **IoT-engine** is responsible for various logics of device message processing.
- **IoT-admin** provides multiple functions such as the product management and the Object model management, for the platform layer.
- **Time-series database** is responsible for the storage and queries of IoT data, which we can choose from candidate tools in market, such as TDEngine, InfluxDB, TimescaleDB, etc.

Details about each module will be introduced later. Note that the Object model is used to describe the property, service, and event associated with an IoT device. Different types of IoT devices are managed with the product as the basic unit, which will be described in details in Section III-B: The Object model.

III. IoTPass Methodology

In this section, we will introduce three concepts related to the architectural design of our system and describe the corresponding advantages. We present microservice architecture in Section III-A to explain how our system efficiently organizes and manages multiple functional modules. Afterward, the Object model which serves as the basis for our database table design and abstraction of IoT related concepts (such as products, devices, etc) is introduced in Section III-B. Then we present Revert-RPC (Remote Procedure Call) communication technique which supports both synchronous and asynchronous service invocation modes in Section III-C.

A. Microservice Architecture

There are two main categories of the IoT architectures named *unified* and *microservice* architectures. Modules in unified architecture are integrated into one unit, which means all of the modules are used at the same time [11]. Using unified architecture has some advantages like uniform standards and independence of modules [12]. However, with the rapid proliferation of IoT devices, overloaded functions will bring bottlenecks to the system under the unified architecture. Moreover, a change in one function will affect other functions due to the function dependencies. Hence, the unified architecture is not suitable for IoT workloads.

IoTPass system adopts the microservice architecture which consists of a series of independent services and each of which is deployed, developed separately, and managed in a distributed manner to achieve the purpose of convenient development and deployment. Multiple researches about the IoT platforms based on the microservice architecture has been done, such as a microservice-oriented big data architecture (MOBDA) which is designed to predict smart transportation [13], and a reactive microservice architecture which is designed for IoT application development [14].

The microservice architecture in IoTPass system is implemented based on Spring Boot [15] and Spring Cloud [16]. Device messages are uniformly accessed and scheduled from the IoT-gateway module; the IoT-admin module and the IoT-engine module are independently deployed by the container and interfaces are called among different modules. Therefore our IoTPass system has the following advantages: modules are registered by communicating with Eureka server which facilitates service expansion; modules are located through service names, and the Feign framework is used to complete remote procedure calls, which is more efficient; requests for multiple identical services are automatically distributed in a balanced manner to avoid excessive single-point overload; targets of the microservice module are monitored through SpringBoot admin monitors, which is convenient for operation and maintenance.

B. Product and The Object Model

IoTPass system manages different devices with the concept of product as the basic unit, and uses the Object model to structure the time-series data. A product is defined a collection of devices, usually a collection of devices with the same functional definition.

The Object model is a data model used to describe the functions of a product and is the digital representation of entities in physical space (such as sensors, vehicle-mounted devices, buildings, factories, etc). It describes what the entity is, what it can do, and what information it can provide externally from three dimensions of property, service, and event.

- **Property**: Properties of a product describe the specific information and status of the device when it is working. For example, the current ambient temperature collected by the environmental monitoring device, the status of the smart light switch, etc.
- **Service**: Services of a product refer to the instruction or method that the device can call externally. Multiple input and output parameters can be set: input parameters are the parameters when the service is executed and the output parameters are the results after the service is executed. Compared to properties, services can implement more complex logic with a single instruction, such as completing a specific task.
- **Event**: Events of a product are the information actively reported to the cloud when the device is working, generally including information, alarms and errors that need to be perceived and processed externally. Events can contain multiple output parameters. For example, the notification information after a specific task is completed; the temperature and time information when the device fails; the working status when the device alarms, etc. In our system, the IoT platform server forwards and pushes events. At the same time, different applications subscribe different events to complete the event distribution and meet the requirements of the microservice architecture.

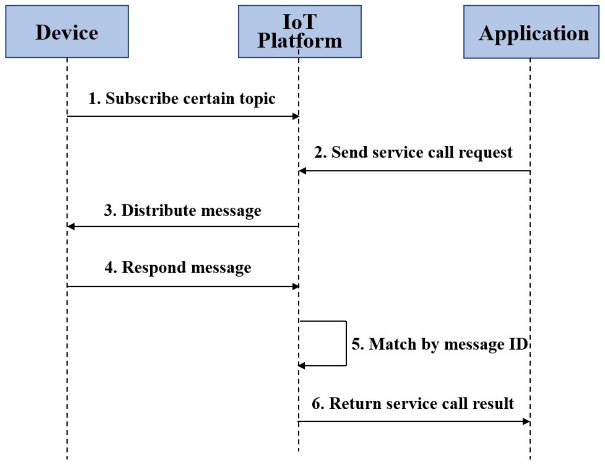

Fig. 2. The workflow of a Revert-RPC Communication (RRPC) service call.

Our system designs each field of the database table based on the Object model, which basically covers the data interaction forms in various usage scenarios of IoT and achieves the purpose of improving the transmission efficiency and reducing the transmission cost.

C. Revert-RPC Communication Technique

RPC (Remote Procedure Call) is a request–response communication protocol that a program can use to request services from another computer program on the network without the programmer explicitly coding the details for the remote interaction. RPC follows a client/server model. However, there are certain downsides to using this model with IoT workloads. Since RPC protocol only supports devices to send service call requests to the server, the server cannot actively request the devices and receive devices' response. Therefore, our system adopts RRPC (Revert-RPC) on the basis of RPC and the workflow of a RRPC service call is shown in Figure 2: the IoT platform provides application programming interfaces to the application, devices only needs to reply the message in a fixed format and the application can use those interfaces to obtain the response result of the device synchronously. At the same time, in order to avoid the performance degradation problem caused by the long response time when the application initiates a time-consuming service call, we also implement the invocation mode of asynchronous service call. Details about two service invocation mode will be introduced in Section IV.

IV. Implementation

In this section, we introduce the implementation details of the architecture of IoTPass system and present the specific functions of each module in details. Firstly, we introduce the communication protocol for data interaction between the device and the IoT platform in Section IV-A. Afterward, we present the specific functions of the IoT-gateway, IoT-engine and IoT-admin modules from Section IV-B to Section IV-D.

Then we introduce the workflow of the two service invocation modes in Section IV-E. The architecture and workflow of our flow access system which ensures high scalability is introduced in Section IV-F.

A. Communication Protocol

IoTPass system supports equipment access using the MQTT standard protocol and the IoT platform uses EMQX as the MQTT broker [17]. As an instant messaging protocol with low overhead and low bandwidth occupation based on TCP/IP, MQTT is more suitable in the scenario where the amount of IoT data is massive and the data is relatively structured and formatted. MQTT uses the publish/subscribe mode for communication and the two communicating parties send and receive messages through Topics. In MQTT, Topic refers to a UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels and each topic level is separated by a forward slash. At the same time, as an open source IoT MQTT message server developed based on Erlang/OTP platform, EMQX's high reliability and high concurrency are also suitable for IoT platform work scenarios.

In our IoTPass system, devices connect to the authentication server through TCP and the authentication server returns the MQTT server information (MQTT IP, port, login user name and necessary information related to security). After one device obtains the relevant information it connects to the MQTT server to complete the communication layer access. The MQTT server decouples the IoT server from the device and provides many-to-many forwarding of data packets during the communication process. Device access to MQTT is equivalent to access to the IoT server.

B. IoT-gateway Module

This module is the access sub-layer of the IoT platform. It is responsible for accessing a large number of devices, processing security-related logic, including message encryption and decryption, access rights, and forwards messages that need to be parsed in the IoT-engine module. The IoT server also sends messages through this module. At the same time, this module also performs simple format conversion, de-duplication and other preliminary operations on the message. In our system, this module supports TCP, HTTP, MQTT and other protocols and has high concurrency capability.

C. IoT-engine Module

This module is responsible for the logic of device message processing and its workflow is mainly divided into three steps.

- **Preliminary processing**: This step specifies the message format based on the Object model. In this step, the message forwarded by the IoT-gateway module is converted into a memory object and the message type is determined for analysis and processing afterwards.
- **Rule engine**: In this step, the IoT-engine module obtains the properties, events and service configuration of the device from the IoT-admin module according to the device type corresponding to the uploaded message and parses the message according to the configuration, extracts data instances with different configurations in the message.Then, these data instances are filtered according to the rules.
- **Message relaying**: In this step, data instances processed by the rule engine are relaid to the time-series database for storage.

D. IoT-admin Module

The main function of this module is to provide related functions of product management and the Object model management, which includes: interfaces for adding, deleting, modifying and querying product's properties, events, and services; device registration and device upgrading. At the same time, this module defines multiple web interfaces for the background web management page, including the interface which returns all historical upload records of the properties of a certain device within the query time interval, the execution status of a certain service call, all historical service call records of the specified service or all services, etc.

E. Synchronous and Asynchronous Service Calls

Our system supports both synchronous and asynchronous service invocation modes. Synchronous service calls refer to service calls with short execution time (generally defined as execution time less than 500 ms), while asynchronous services are service calls with long execution time. The workflow of these two service invocation modes is as follows.

1) Synchronous Service Call: In this case, the application initiates a request to the IoT platform and invokes the service call of a specific device. After receiving the request, the IoT platform performs a series of steps such as numbering, depositing and entering the cache, then publishes action instructions through MQTT. After receiving the command, the device executes the action and returns the execution result (including failure situations). At the same time, the application directly waits for the result synchronously. Generally, the IoT platform returns the service result in one second. If the device does not reply to the message within two seconds, the IoT platform is required to respond to the application with a timeout response. The result of the timeout will also be saved in the database for future reference.

2) Asynchronous Service Call: The process of request sending and relaying to the device is the same as the Synchronous case. After receiving the command, the device immediately replies to the in-progress message and marks it as asynchronous service call in the data packet. After the IoT platform receiving this message, it resets the timeout to 30 seconds and replies to the application the server-related information (asynchronous tag and command ID number), then ending the application's invocation process. The IoT platform will continue to maintain the command execution status and wait for the device execution status to be updated (with the command ID number for matching). The device is required to upload the command execution heartbeat information every

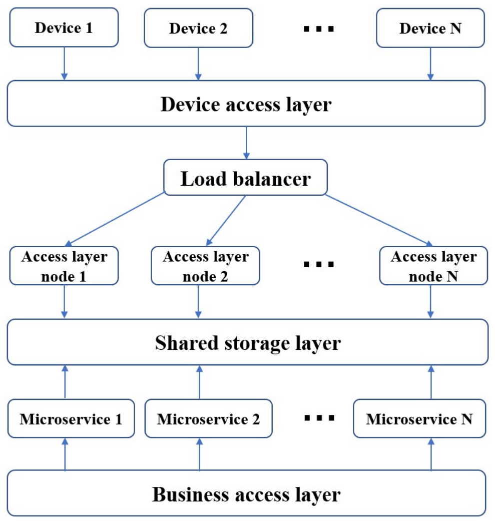

Fig. 3. The architecture of flow access system, which can be completed through two access layers (device access layer and business access layer).

25 seconds to confirm that the device is still in progress. If the execution report is not received after the timeout, it is regarded as an abnormal end and if the execution is completed, the execution result is reported, and the regular command execution heartbeat uploading is terminated. The results of both cases will also be saved in the database. The application can subscribe events or query to the IoT platform to get the results. The handling of the timeout logic for not receiving the device's response is the same as for the synchronous service call.

F. High Scalability

In order to achieve high scalability, our system adopts the idea of separation of storage and access. The architecture of our flow access system is shown in Figure 3. We complete the access of flow through two access layers.

- **Device access layer**: This layer is accessed through the MQTT protocol as mentioned before. A device is randomly connected to the back-end access layer node through the front load balancing module. The access layer node maintains the mapping relationship of the client ID to the certain TCP connection. At the same time, after any device is connected the messages are relayed to the shared storage layer. The shared storage layer maintains mapping relationship of the client ID to its access node and the global Topic tree (implemented by a Trie tree).
- **Business access layer**: This layer corresponds to the applications and is used to access massive service requests sent from the users. After the flow arrives, this layer will first send a query to the shared storage layer about which broker the device is connected to and then complete the delivery correctly. The query process is to determine which clients subscribe to this Topic through the Topic tree firstly and then determine the access layer node through the client ID.

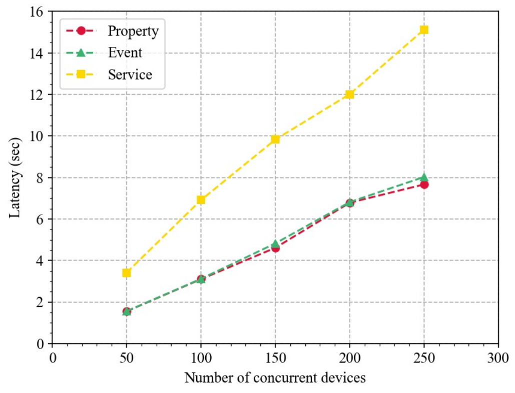

Fig. 4. Response latency of three communication forms, namely property uploading, event uploading and service call.

Since our business access layer adopts the microservice architecture (mentioned above), it naturally supports high scalability. And due to the existence of shared storage, the device access layer is stateless and can be extended horizontally: if one access layer node fails, it can be re-enabled on a new access layer node without affecting the normal routing of messages (routing information is stored in shared storage). Then we use the Etcd solution to ensure the high availability of the shared storage layer. Etcd is a strongly consistent, distributed key-value store that provides a reliable way to store data that needs to be accessed by a distributed system or cluster of machines [18]. Using the above design, IoTPass achieves high scalability of the entire flow access system, making it suitable for large-scale equipment clusters.

V. Evaluation

In this section, we present the experimental results of IoTPass system.

A. Setup

We choose TDengine [19] as the database, which is a high-performance, scalable time-series database with SQL support. Since it is optimized for the characteristics of time-series data, our system could efficiently store the massive amount of IoT data with our database table design based on the Object model. We use a Python script to simulate the communication between devices and the IoT platform (through MQTT connection) and between applications and the IoT platform (through HTTP request). We use response latency as the evaluation metric, since it well reflects the data processing capabilities of the IoT platform as a medium for coordinating the communications from applications to devices.

B. Platform-device Communication Benchmark

We first evaluate the performance of the communication between devices and the IoT platform, which is the most frequently used in IoT workloads. Since we adopt the Object

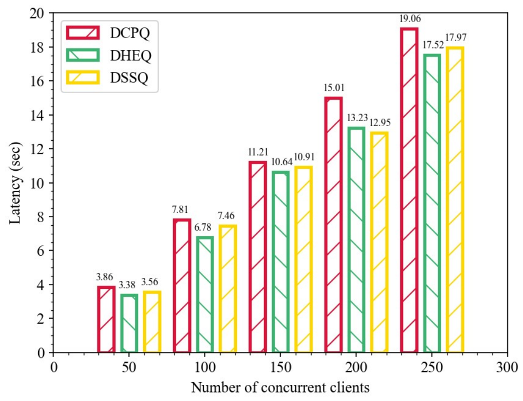

Fig. 5. Latency of three web interfaces provided by IoT-admin module: Device Current Property Query (DCPQ), Device History Event Query (DHEQ), and Device Service State Query (DSSQ) .

model to abstract the workflow of IoT, the communication between devices and the IoT platform is divided into three forms: property uploading, event uploading and service call. We experimented three forms of communication and tested the latency from sending messages from the device to receiving, parsing, and storing messages on the IoT platform with different numbers of devices connected to the IoT platform simultaneously.

As shown in Figure 4, the response latency of the property uploading process and event uploading process is similar (since their message contents are similar). The service call requires two-way communication between the device and the IoT platform, hence the response latency is higher than the previous two. At the same time, due to our flow access system, the IoT platform has not experienced a sharp increase in response latency when the number of concurrent devices increases, providing the possibility for future cluster expansion.

C. Microservice Architecture Benchmark

The performance of the implemented microservice architecture is evaluated through three web interfaces provided by IoT-admin module: (1) **Device Current Property Query (DCPQ)**: This interface returns the basic information of the queried device and its current properties. The current properties refers to all properties after the last upload from the device. (2) **Device History Event Query (DHEQ)**: This interface returns all historical upload records of the events of the queried device within the query time interval. (3) **Device Service State Query (DSSQ)**: This interface returns the current execution status of a specific service call, including success, processing, failure, etc.

We tested the latency from sending HTTP requests to the IoT platform, to receiving HTTP responses with different numbers of clients. As shown in Figure 5, the latency of DHEQ interface is higher than the remaining two with any number of clients. That is because DHEQ interface needs to query more records in the time-series database, hence the query step takes a longer time. Owing to our flow access system and microservice architecture, our business access layer also has not experienced a sharp increase in latency in the case of an increase on the number of concurrent devices.

VI. Conclusion

In this paper, we present, IoTPass, an IoT data management system for processing time-series database, which adopts microservice architecture, the Object model and Revert-RPC communication technique to improve performance and scalability. It can effectively receive, process, relay and store massive amount of IoT data, providing a basis for subsequent data analysis. Experiment results demonstrate the effectiveness and scalability of our system.

Acknowledgment

This project is partially sponsored by National Key Research and Development Program No. 2020YFB2103801, the 13th 5-year plan Civil Aerospace Technology Advance Research Project (D030301), NSFC 62032003, and BJNSF L192004.

References

[1] Perumal T, Sulaiman M N, Sharif K Y, et al. Development of an embedded smart home management scheme[J]. International Journal of Smart Home, 2013, 7(2): 15-26.
[2] Li S, Wang H, Xu T, et al. Application study on internet of things in environment protection field[M]//Informatics in Control, Automation and Robotics. Springer, Berlin, Heidelberg, 2011: 99-106.
[3] Chang H, Zhou N, Zhao X, et al. A new agriculture monitoring system based on WSNs[C]//2014 12th International Conference on Signal Processing (ICSP). IEEE, 2014: 1755-1760.
[4] Emerald, ”That ‘Internet of Things’ Thing,” [Online]. Available: https://www.rfidjournal.com/that-internet-of-things-thing.
[5] Cboe, ”Market History Monthly - 2021,” [Online]. Available: https://www.cboe.com/us/equities/market_statistics/historical_market_volume/market_history_monthly_2021.csv-dl
[6] Raman V, Attaluri G, Barber R, et al. DB2 with BLU acceleration: So much more than just a column store[J]. Proceedings of the VLDB Endowment, 2013, 6(11): 1080-1091.
[7] Lamb A, Fuller M, Varadarajan R, et al. The vertica analytic database: C-store 7 years later[J]. arXiv preprint arXiv:1208.4173, 2012.
[8] ”InfluxDB, Time-Series Database—Even Better in the Cloud” [Online]. Available: https://www.influxdata.com.
[9] ”Timescale: Time-series data simplified,” [Online]. Available: https://www.timescale.com.
[10] Wang C, Huang X, Qiao J, et al. Apache IoTDB: time-series database for internet of things[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2901-2904.
[11] Dmitry N, Manfred S S. On micro-services architecture[J]. International Journal of Open Information Technologies, 2014, 2(9): 24-27.
[12] Parvizimosaed M, Noei M, Yalpanian M, et al. A containerized integrated fast IoT platform for low energy power management[C]//2021 7th International Conference on Web Research (ICWR). IEEE, 2021: 318-322.
[13] Santana C, Andrade L, Delicato F C, et al. Increasing the availability of IoT applications with reactive microservices[J]. Service Oriented Computing and Applications, 2021, 15(2): 109-126.
[14] Razzaq A. A systematic review on software architectures for iot systems and future direction to the adoption of microservices architecture[J]. SN Computer Science, 2020, 1(6): 1-30.
[15] “Spring Boot,” [Online]. Available: https://spring.io/projects/spring-boot.
[16] ”Spring Cloud,” [Online]. Available: https://spring.io/projects/spring-cloud.
[17] ”EMQX: The World’s #1 Open Source Distributed MQTT Broker,” [Online]. Available: https://www.emqx.io.
[18] ”etcd,” [Online]. Available: https://etcd.io.
[19] ”TDengine: Time Series Database An Open Source TSDB,” [Online]. Available: https://tdengine.com.