DAQ-Middleware: Data Acquisition Middleware based on Internet of Things

Zhijin Qiu*†, Zhongwen Guo*, Shuai Guo*†, Yingjian Liu* and Yu Wang†‡
* Ocean University of China, Qingdao, Shandong, China
† University of North Carolina at Charlotte, Charlotte, North Carolina, USA
‡ Taiyuan University of Technology, Taiyuan, Shanxi, China
Email: qzjouc@163.com, guozhw@ouc.edu.cn, guoshuaiouc@163.com, liuyj@ouc.edu.cn, yu.wang@uncc.edu

***Abstract*—With envisioned Industry 4.0, the Internet of Things (IoT) has been widely used in many fields. Unfortunately, the format, type, and access methods of data sources in different areas are diverse, and this makes upper development complex and low efficiency of data acquisition. To address this issue, a novel scalable data acquisition middleware (DAQ-Middleware) is proposed in this paper. Through the design of the architecture model, and the standardization of access methods and interfaces, DAQ-Middleware implements fast access to heterogeneous data sources and provides standardized formatted sensing data for IoT applications. Furthermore, in order to improve the rate of sensor data acquisition, we also propose a parallel data acquisition algorithm and an acquisition efficiency optimization heuristic method. Via the development and application of proposed DAQ-Middleware, this middleware has been validated in the aspect of the rationality, feasibility, development efficiency and data acquisition efficiency. Our results confirm that DAQ-Middleware can achieve the logical isolation between the data sources and the IoT application systems, satisfy the automatic fast access to the heterogeneous data sources, reduce development cost, and enhance data acquisition efficiency.**

I. Introduction

With the development of pervasive computing, RFID, and sensor networks, the application and development of IoT technologies have been promoted. In different areas, IoT through the connection of intelligent devices achieves the global sensing data acquisition, storage, analysis, and display. A typical IoT system is a large, dynamic, and scalable sensor network that enables data identification and acquisition from intelligent devices, RFID tags, computers, and mobile devices [1]. These devices generate large amounts of real-time-serialized data accessing to the IoT through a high-speed network. At the same time, the data sources also include the existing databases and data files, which can realize the data integration of the existing IoT systems. Therefore, the IoT data sources include two parts: 1) the data obtained by the sensing devices; 2) the existing databases and data files. At present, no matter what kind of data sources, there are always diversity issues, such as different data acquisition units, data types, communication protocols, access methods, and so on are used by different companies. The developers have to develop various data acquisition systems for different combinations of accessing sensor devices, and this issue makes the development of IoT systems complicated. Meanwhile, different data acquisition systems are designed to be used in different scenarios, so they cannot simultaneously obtain data for multiple data sources. Performing data acquisition sequentially results in lower efficiency. Obviously, it is alway challenging to make different instrument manufacturers and existing databases and data files to use a unified communication protocol and data format.

In terms of IoT system design, many related studies have been conducted. Most of them mainly focus on the overall architecture design [2]–[4] and data display method [5]–[8] in IoT systems. There is no much study about the diversity of the data sources' format. In the research area of distributed systems, the relevant scholars have designed different architectures or algorithms for data acquisition system. Kovac [9] propose the use of virtual instrument technology and GPIB interface to achieve the acquisition of sensor data, which increases the convenience of access to the sensor devices to a certain extent. Qiu et al. [10] propose a high performance data acquisition algorithm based on the analysis of dynamic delay characteristics of data acquisition. But this algorithm is only suitable for specific sensing devices, and does not apply to multiple data sources. Until recently, Qiu et al. [11] and Hu et al. [12] propose the concept of Complex Virtual Instrument System to handle multiple data sources. However, their architectures and application areas are still restricted. The Open GIS Consortium (OGC) [13] proposes to use Programmable Underwater Connector with Knowledge (PUCK) protocol to integrate the physical devices automatically. The system uses a computer to put the configuration information into PUCK model, and connects this model to the physical ocean observing instruments, which can realize automatic instruments access. Although this has solved lots of problems in system integration and development, it must modify the ocean observing instruments and add PUCK model. Doing so increases the cost of instrument manufactures as well. In summary, none of the solutions above can settle comprehensively all the mentioned problems in data acquisition of IoT systems.

Though the requirements of data analysis and display are very different in different IoT systems, their data acquisition methods are not vary greatly depending on the application domain. In this paper, according to the experiences of participation in the development of IEEE 1851 and IEEE 2402 international standard, we propose a data acquisition middleware based on IoT: DAQ-Middleware. DAQ-Middleware

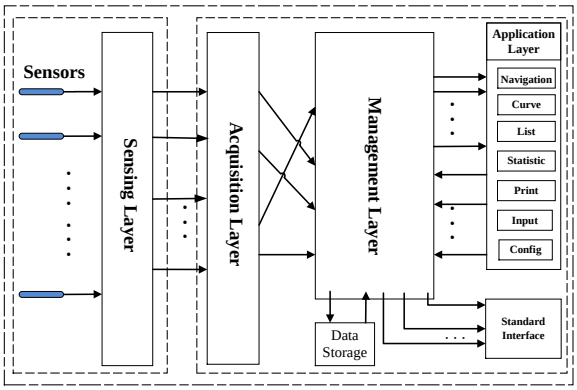


Fig. 1: Hierarchy model of IoT.

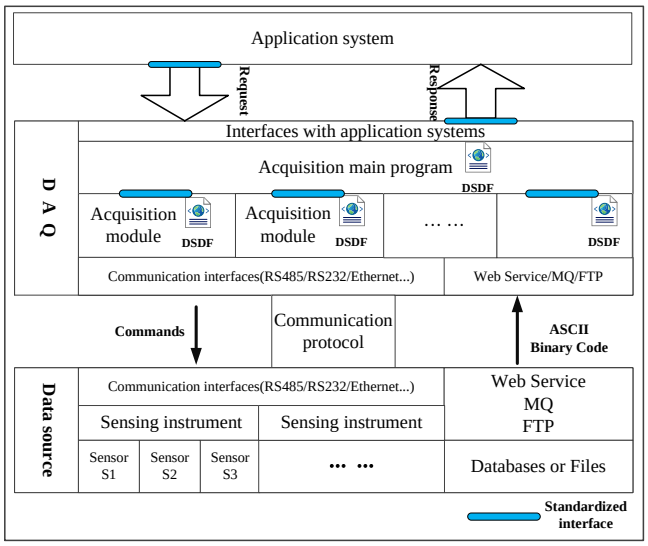


Fig. 2: Architecture of DAQ-Middleware.

is located between sensing devices and operating system. DAQ-Middleware not only can improve the efficiency of data acquisition (using parallel data acquisition algorithm and efficiency optimization method), but also achieve an automatic access to variety of heterogeneous data sources in IoT system. DAQ-Middleware can provides sensing data for IoT system of the different areas through the standardized data interface. In summary, the major contributions of this paper can be summarized as follows.

- We design a scalable IoT data acquisition middleware, DAQ-Middleware, which supports not only the sensing device's interfaces for the serial port, network port, GPIB and USB, but also the interfaces of various data type databases and data files for FTP, Web Service and MQ.
- Through the architecture design, DAQ-Middleware achieves a new data source for automatic access, support for dynamic addition of front-end data sources without re-programming. Meanwhile, a unified definition of the data description format hides semantic description in the hardware level, so that it is easier for developers to development systems.
- DAQ-Middleware uses a newly designed parallel data acquisition algorithm and an efficiency optimization heuristic to improve the data acquisition efficiency, system stability, and data accuracy.
- The standard back-end interfaces of DAQ-Middleware are defined, including the control command interface, the data request interface and the data return interface, which can facilitate IoT integration.

The rest of this article is organized as follows. Section 2 presents the architecture model of DAQ-Middleware. Section 3 provides the detailed design of DAQ-Middleware. Section 4 shows the application development cases with DAQ-Middleware and the analysis of its performance. Finally, conclusions are proposed together with an indication of directions, which deserve further work, in Section 5.

II. Architecture Model

A. IoT Hierarchy Model

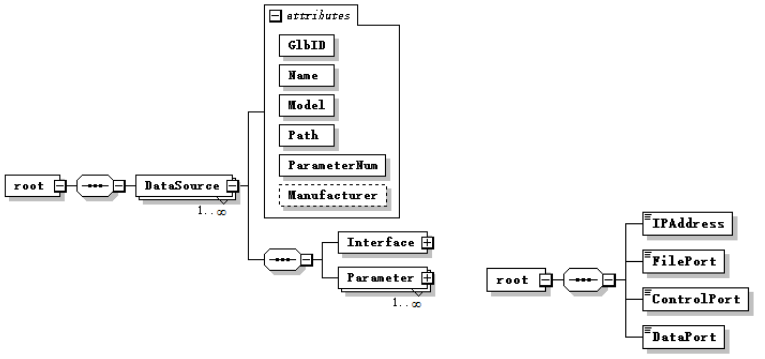
Recently, many scholars put forward the object hierarchy model of IoT according to the different requirements, in order to facilitate the development and maintenance of IoT systems. Even though different hierarchies of IoT has been proposed, the main functional modules are similar. According to [6], the object hierarchy model is divided into four layers (sensing layer, acquisition layer, management layer, and application layer), as shown in Figure 1. Each layer is independent of each other, and only through the data interfaces to interact between the adjacent layers. The output of a layer is the input of next level.

- **Sensing Layer:** It is the IoT data sources, generally refers to sensing devices.
- **Acquisition Layer:** It gets sensing data from sensing layer, and transfers data to management layer according to the standard data formats.
- **Management Layer:** It implements the data processing and storage. At the same time, it provides standardized data for other systems via the standardized data interfaces.
- **Application Layer:** It visualizes and displays the acquired data. Its functions include data list, curve, navigation, statistical analysis, printing, and so on.

In some cases, management and application layers are collectively called application system.

B. DAQ-Middleware Architecture Model

Architecture model of DAQ-Middleware is shown in Figure 2. It is located in the acquisition layer of the IoT hierarchy model. Its functions include receiving control commands from the application system, periodically obtaining the data of data sources, and transmitting the obtained data to the application system through the standardized data interfaces. DAQ-Middleware is the middle part of the computer operating



(a) DSDF (b) CPDF

Fig. 3: DSDF and CPDF structures in XML schema.

system connected to the data sources. DAQ-Middleware uses Socket to communicate with application system. In different circumstances, DAQ-Middleware connects to the sensing devices via interfaces such as RS232, RS485, Ethernet and USB, and connects to the existing databases and data files through another interfaces, such as Web Service, MQ and FTP. The major works of this paper are to design the middleware structure model, propose parallel data acquisition algorithm and efficiency optimization heuristic, and realize the automatic access to different data sources, with goals of reducing the development cost and improving the data acquisition efficiency.

III. DAQ-MIDDLEWARE

In this section, we first define the data source description file, the interfaces between the application system and DAQ-Middleware, the interactive mode between the main program and the acquisition modules. Then, we propose the parallel data acquisition algorithm and the efficiency optimization heuristic. Last, we describe the overall process of data acquisition in DAQ-Middleware.

A. Data Source Description File

The data source description file (DSDF) describes all the information of the data sources that are accessed. DAQ-Middleware can obtain Various information, such as attributes, interfaces and sensor parameters of the data sources by analyzing DSDF. In order to facilitate the DSDF analysis, the format of DSDF is defined.

DSDF is described in XML format. As show in Figure 3a, DSDF contains multiple data sources, and each data source includes information of attributes, interfaces and parameters. The node of attributes holds information such as serial number (GlbID), the name of data sources (Name), the name of acquisition module (Model), acquisition module's storage path (Path), the number of sensor parameters (ParameterNum), and data sources' provider (Manufacturer) which is optional.

DSDF describes the ways of access, and the interface parameter's information of each data source, so that we can achieve the connection from DAQ-Middleware to the data sources. Different data sources are connected uses different

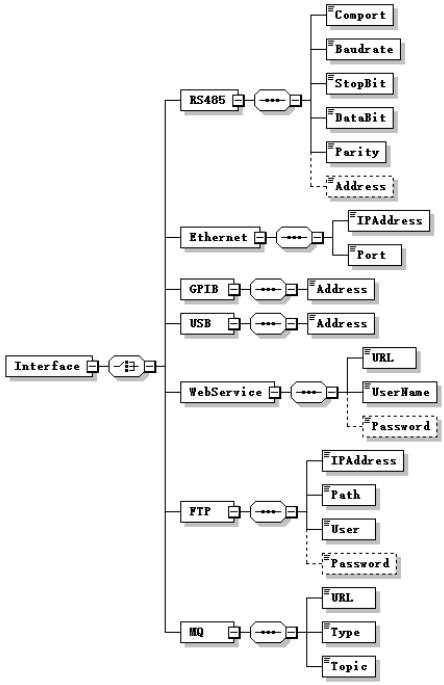


Fig. 4: Description structure of interface node.

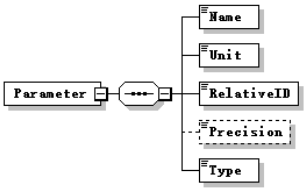


Fig. 5: Description structure of parameter node.

interfaces, and the parameters of different data interfaces are not the same. As shown in Figure 4, the information description formats of different interfaces are defined in DSDF. The description of the interface information includes the RS-485 (or RS-232), Ethernet, GPIB, USB, Web Service, FTP and MQ. Different types of interfaces need to describe the parameters are different. For example, RS-232 needs to describe the information of serial number, baud rate, data bits, stop bits, and checksum, but RS-485 also includes the information of sensing device's address.

DSDF also describes the sensing parameters, which contained in each data source. DAQ-Middleware obtains the parameter information and realizes the acquisition of the sensing parameters by analyzing the description information of sensing parameters in DSDF. As shown in Figure 5, the node of parameter includes sensor parameter name (Name), unit (Unit), relative ID (RelativeID), precision (Precision) and parameter's type (Type). The sensing parameters in different data sources are numbered separately, so the RelativeID refers to the ID relative to the data source. DAQ-Middleware obtains the corresponding sensor parameters by obtain GlbID and RelativeID. Precision refers to the position of the decimal

point. Type includes both analog and binary types.

In summary, DSDF contains all the necessary information for the middleware to achieve data acquisition. Through the standardization of DSDF, different data source information is standardized, and it is easy to parsed by DAQ-Middleware.

B. Interface Standardization

DAQ-Middleware is an independent system. So if you want to achieve no matter which the field of IoT can access to DAQ-Middleware, we need to standardize the interfaces between DAQ-Middleware and the application system. All these interfaces adopting Socket, which include three parts: data source description file transfer interface, status control interface, and data acquisition interface. The communication parameter description file (CPDF) is used to describe the interface information of DAQ-Middleware. By analyzing CPDF, we can realize the conjunction between the application system and DAQ-Middleware. In Figure 3b, CPDF structure in XML schema is described.

IPAddress indicates the IP address of the computer, on which the DAQ-Middleware was deployed. The application system establishes a connection with DAQ-Middleware through this IP address. When the connection is established for the first time, DAQ-Middleware transfers the DSDF to the application system through the FilePort interface. The application system sends the control commands to DAQ-Middleware through the ControlPort interface. Then DAQ-Middleware executes these commands and returns the setting results to the application system.

DataPort is defined as the interface between the application system and DAQ-Middleware to request and return sensor data. In particular, DSDF is transmitted using TCP/IP through DataPort. In this paper, the formats of request and the corresponding command are standardized between the application system and DAQ-Middleware, in order to achieve control of the middleware and acquire of sensor data.

Application system controls the start and termination of DAQ-Middleware through ControlPort. The format of request commands is shown in Table 1. This command consists of six parts: ① command beginning character; ② APC* represents this command as a status control command; ③ the function of the command, where APStart and APStop represent start and stop data acquisition, respectively; ④ separation character; ⑤ check code, which is the sum of all the characters (in decimal code) of the first three parts in a 256 modulus; ⑥ a termination symbol $. The format of ControlPort's response command is shown in Table 2. This command consists of nine parts. Many of them are similar to those in request commands, except for ② R means that this command is a status control response command; ⑥ Result indicates whether the command was successful (1 indicates that the operation has successful, and 0 otherwise).

The application system obtains the sensor data, which acquired by DAQ-Middleware through data acquisition interfaces. The sensor data consists of analog value and status variables. Table 3 shows the request command's format for DataPort, which consists of six parts. ② DR* indicates the request command; ③ is the requesting sensor list, whose format is '$SensorID$, $SensorID$,$\cdots$'. N (size of the list in Bytes) is no more than 65535. Table 4 shows the format of response command for DataPort, which consists of eight parts. ② D represents that this command is a data acquisition command. ③ Standard data format is '*yyyy-MM-dd HH:mm:ss.fff*', which indicates year-month-day hour:minute:second.millisecond. ⑤ DataList is the sensor data matched with the sensor ID, in the format of '$SensorID, value@SensorID, value@\cdots$', where '$value$' is decimal data. If the data is invalid, '$value$' would be NULL.

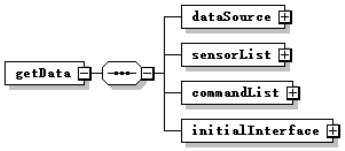


Fig. 6: getData function.

C. Acquisition Module

Figure 2 shows that DAQ-Middleware includes the acquisition main program and the acquisition module. DAQ-Middleware uses communication interfaces to acquire the data from the data sources based on specific protocol. Different sources' interfaces have different communication protocols, so we have developed a unique acquisition module for each data source based on different communication protocols. The acquisition module can be a jar file or a separate DLL file.

The functions of acquisition main program include the following. It accepts control commands and data request commands from the application system and returns DSDF and the processing results. It is responsible for obtaining DSDF corresponding to each acquisition module, and synthesizing-analyzing the information contained in DSDF to implement the loading of the acquisition module. It instantiates the communication interface, which is described in the DSDF. It interacts with the acquisition module, organizes the obtained sensor data in a standard format, and returns the sensor data to the application system.

The functions of acquisition module include the following. It interacts with the acquisition main program through the standard interface, obtains the sensor sequence which needs to acquire by the acquisition main program, and returns the sensor data to the acquisition main program. It obtains sensor data from data source via communication protocol, and the information of communication interfaces, which have been instantiated by the acquisition main program.

The standardization of interface between the acquisition main program and the acquisition module enables the dynamic modification and deletion of the data source by modifying DSDF. This interface is implemented using the getData function, which is encapsulated in the acquisition module. As Figure 6 shows, the parameters of getData function include data source description information (datasourceInfo), sensors

TABLE I: Format of ControlPort's request command.

ID	①	②	③	④	⑤	⑥
Command	#	APC*	APStart/APStop	&	Check code	$
Size in Bytes	1	4	7	1	2	1

TABLE II: Format of ControlPort's response command.

ID	①	②	③	④	⑤	⑥	⑦	⑧	⑨
Command	#	R	APC*	APStart/APStop	&	Result	&	Check code	$
Size in Bytes	1	1	4	7	1	1	1	2	1

TABLE III: Format of DataPort's request command.

ID	①	②	③	④	⑤	⑥
Command	#	DR*	SensorList	&	Check code	$
Size in Bytes	1	3	N	1	2	1

TABLE IV: Format of DataPort's response command.

ID	①	②	③	④	⑤	⑥	⑦	⑧
Command	#	D	yyyy-MM-dd HH:mm:ss.fff	&	DataList	&	Check code	$
Size in Bytes	1	1	23	1	N	1	2	1

list (sensorList) and instantiated interface information (instantiatedInterface).

The format of data source description information is the same as the definition in Figure 3a, but the data source description information passed by the getData function only contains the information of a data source that needs to be acquired. The sensorList is the list of sensors' relative ID. Multiple IDs separated by ','. This ID is consistent with the one in the DSDF. To achieve data acquisition, acquisition module directly calls the interface that has been instantiated by the main program. Because the instantiated content are different with various communication interfaces, the instantiated descriptions of all supported communication interfaces are standardized. In the case of serial communication, the parameters that need to be passed after instantiation include the port identifier (portID), port (serialPort), the input stream (inputStream), the output stream (outputStream), and so on.

The getData function returns a data of String '$time$: $content$\$$state$'. Here, '$content$' is the returned sensor data. The formats of '$time$' and '$content$' are '*yyyy-MM-dd HH: mm: ss.fff*', and '*SensorID,Value@SensorID,Value@*$\cdots$', respectively. State is the state of current communication (1 is normal, 0 otherwise).

By defining the getData function and developing the corresponding data acquisition module according to each data source communication protocol, the acquisition main program dynamically loads data acquisition module through the reflection mechanism base on the data acquisition module placement path. By reading the information of 'path' and 'model' in the attribute node of DSDF, the deployment path of the data acquisition module is obtained.

When adding, modifying and deleting data sources using the method of main program isolate from the acquisition module, we do not need to do secondary code development. This method improves the flexibility, scalability and efficiency of DAQ-Middleware.

D. Parallel Data Acquisition

As Figure 2 shown, the main program receives data request commands from application system, which contain the sensor list of multiple sensing parameters for multiple data sources. In general, the data acquisition middleware analyzes the sensor list, obtains the data sources to be acquired, and then performs the data acquisition *in turn*. However, with the increasing number of data sources, this method leads to long acquisition cycle and low efficiency. Since the characteristics of data acquisition module are independently designed in the DAQ-Middleware, we propose a novel parallel data acquisition algorithm, aiming to improve the efficiency of data acquisition.

According to the principle of computer interfaces, there is a situation in which the same interface can access multiple data sources. For example, an RS-485 interface can access multiple sensing devices (with different device addresses). But in order to accurately analyze the returned sensor data, every time the data acquisition middleware can only communicate with a single device on the same interface. Since data acquisition between different data sources is independent of each other, we can perform them in parallel.

Let I_i denote interface information, where i is the index of interface and $i = 1, 2, \cdots m$. The data source information is defined as R_j and j is the index of data source for each interface connection. The number of data sources connected to the I_i interface is $n_i^{'}$, i.e., $j = 1, 2, \cdots n_i^{'}$. As shown in Figure 7, we define a matrix D_{m*n}, which represents the data source information of each accessed interface, obviously, $n = max(n_i^{'})$, $i = 1, 2, \cdots m$. Because different data interfaces are

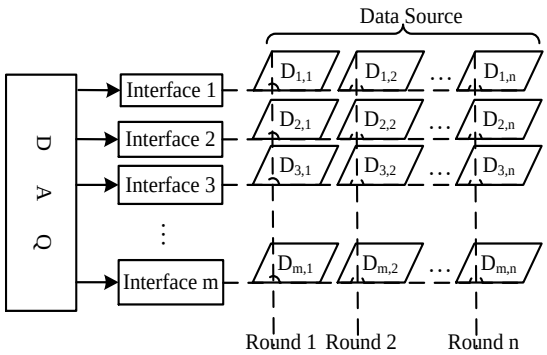


Fig. 7: Parallel data acquisition.

Algorithm 1: Parallel Data Acquisition Algorithm

Input: Matrix D_{m*n} obtained from DSDF
Output: Acquired sensor data and acquisition time T_{m*n}

```
for round j = 1 to n do
    for interface i = 1 to m do
        if D_{i,j} ≠ 0 then
            Acquisition main program sends the sensor
            list to the acquisition module corresponding
            to data source D_{i,j};
        end
    end
    for interface i = 1 to m do
        if D_{i,j} ≠ 0 then
            Acquisition modules collect data according
            to the communication protocol and record
            the time T_{i,j} required for data acquisition;
        end
        if D_{i,j} = 0 then
            T_{i,j} = 0;
        end
    end
end
```

independent of each other, we divide the data acquisition into n rounds, and each round we collect data from a data source on m interfaces. Note that there is a possible situation where no data source corresponding to the elements of matrix $D_{i,j}$, so we set that value 0. Otherwise, we set the value to the data source's ID. Our parallel data acquisition algorithm is described in detail as Algorithm 1.

Parallel data acquisition algorithm can greatly improve the efficiency of data acquisition. According to the obtained acquisition time T_{m*n}, we can calculate the time required to complete one round of data acquisition. The time required for each round of data acquisition is the maximum value of each column of matrix T_{m*n}, namely $t_j = \max_i T_{i,j}$, where $i = 1, 2, \cdots m$. The time t required to complete a complete data acquisition is $t = \sum_{j=1}^{n}(t_j) = \sum_{j=1}^{n}(\max_i T_{i,j})$. If the traditional sequential data acquisition mode is used, the time $t^{'}$ required is $t^{'} = \sum_{i=1}^{m}\sum_{j=1}^{n}(T_{i,j})$. Obviously, $t \leq t^{'}$, and the efficiency of data acquisition is improved significantly with the increasing of the number of data sources.

E. Acquisition Efficiency Optimization

Since different types of interfaces have different characteristics and constraints, there are different allocation methods when we allocate devices and interfaces. In order to improve the efficiency of data acquisition, we can adjust the matrix D_{m*n} to reduce the total acquisition time t, while satisfying the constraints of interface attribute. This optimization problem can be described as follows.

$$\min \quad t = \sum_{j=1}^{n}(t_j) = \sum_{j=1}^{n}(\max_i(T_{i,j}))$$
$$s.t. \quad I_i \textit{ is satisfied}$$
$$\textit{Each data source only belongs to one interface.}$$

We can find the optimal distribution using brute-force method, which goes through all possible allocations of data resources. With n data resources and m possible interfaces per data resource, there are m^n allocations. Thus, brute-force method has an exponential time complexity. Instead, we now propose a heuristic method, which can obtain a reasonable data resource and interface matching efficiently.

Because the data sources are divided into parallel multi-rounds, and each round of data acquisition time is decided by $t_j = \max_i(T_{i,j})$. The basic idea of heuristic is to put the data sources of different interfaces with similar data acquisition time in the same round to save data acquisition time. The detailed algorithm is given in Algorithm 2. Firstly, there are often circumstances that the interfaces with different IDs belong to the same interface type, so we consolidate data sources of the same type of interface in T_{m*n} to obtain $G^{'}_{p*q}$. p is the number of interface types and q is the number of data sources owned by that type interfaces. Obviously, $p \leq m$ and $q \geq n$. The data source communication time for each interface type is then sorted in descending order to obtain G_{p*q}. Next, keep the columns of $D_{i,j}$ unchanged, and the rows gradually increasing. According to the interface I_i and matrix G_{p*q}, we assign data sources to the interface in turn, and then move to the next column. Repeat this until all the data sources are completed. Lastly, the relationship matrix $D_{i,j}$ is obtained between the data sources and the interfaces.

After Algorithm 1, based on $D_{i,j}$, the relationship is established between the data sources and the interfaces. Data acquisition can then be performed according to the parallel data acquisition algorithm.

F. Interaction Process

Figure 8 shows the interaction process among application system, DAQ-Middleware (acquisition main program and acquisition module) and data sources. The interface definitions between application system and acquisition main program and those between acquisition main program and acquisition module are described in Section 3.B and 3.C, respectively.

Algorithm 2: Acq-Efficiency Optimization Heuristic

Input: I_i, T_{m*n}
Output: $D_{i,j}$

```
Consolidate data sources of the same type interface and get G'_{p*q};
for each number of interface types do
    Sort data source communication time in descending order in G'_{p*q} to get G_{p*q};
end
for each round j do
    for each interface i do
        if data source with interface type i of G_{p*q} has not been allocated then
            D_{i,j}=ID of the data source;
        else
            D_{i,j} = 0;
        end
    end
end
```

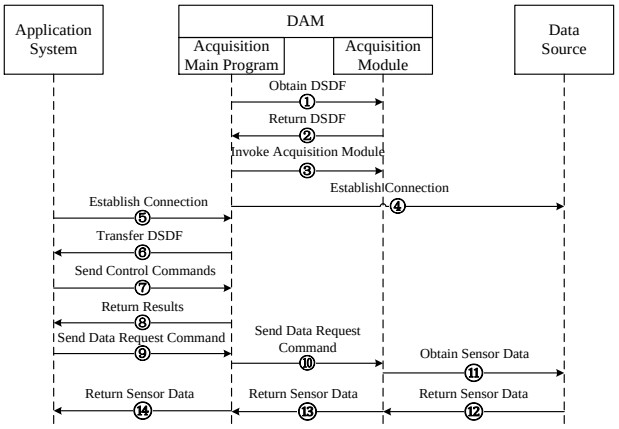


Fig. 8: Interaction process of DAQ-Middleware

Basically, DAQ-Middleware provides a standardized data interface, which can be achieved via application systems docked with DAQ-Middleware in any IoT field. We only need to keep the application system and middleware in the same LAN, configure the application system according to the CPDF, and then complete the physical connection between the system and DAQ-Middleware. The system then can obtain the DSDF, control the DAQ-Middleware, and get sensor data according to the defined standardized interfaces. This docking operation between application system and DAQ-Middleware becomes very simple and convenient.

IV. Implementation and Evaluation

In order to analyze the performance of DAQ-Middleware, several IoT application systems of household appliances testing are designed and developed. The feasibility, development efficiency, and data acquisition efficiency of DAQ-Middleware are analyzed and verified.

Fig. 9: The testing scenario of appliances.

A. Experimental Environment

The field of household appliances testing has a wide range of needs for IoT application systems. In a typical application scenario as shown in Figure 9, a large number of sensors are deployed, and hundreds of operational parameters of appliances are obtained. This application system obtains real-time running information of appliances by DAQ-Middleware. Through analyzing the real-time situation of appliances, manufacturers design production strategy to improve the product quality.

Table 5 summarizes the 9 data sources we used, which consist of sensing devices, databases and data files, 40 devices, and 410 parameters. Each data source corresponds to a data acquisition module, so we have developed a total of 9 acquisition modules, and configured each DSDF. Each data source is connected to DAQ-Middleware through a high-speed monitoring network. DAQ-Middleware provides time-series data for the collection to the IoT application system with interval of 1s. We have also developed the IoT application system to achieve the data processing, analysis, and visual display. Through the system development and verification, it proves that the DAQ-Middleware structure model design is feasible, and the standardization of the interface is reasonable and practical.

B. Development Efficiency

There are many development methods, such as coding, modification and component reusing. Usually, learning a new object-oriented programming language costs lots of time. Therefore the configuration tool of description file is developed, which makes the quick configuration of DSDF easier. According to our model, our configuration tool can lead the developer to finish the configuration procedures without writing any code. All operations are done in the form of a dialog box, so there is no need for the user to understand any programming language.

We take the household appliances testing system as an example. During the procedure of system development, it is further confirmed that our configuration tool is superior in the aspect of improving development efficiency. We compare our configuration tool with three other methods, including coding, modification, and component reusing. As shown in Figure 10, through the development time statistics of the

TABLE V: Information of data sources.

Data sources: Sensor Devices (Interface types: RS-232, RS-485, USB, Ethernet)					
Nmaew	Function	Number of Parameters	Number of Devices	Total of Parameters	ID
MX100	Acquisition Temperature	D	5	300	1-5
SR93	Temperature Controller	2	4	8	6-9
8775A	Power Meter	5	4	20	10-13
UT35A	Indicating Controller	4	4	16	14-17
Anemometer	Measuring Winds	2	5	10	18-22
Data sources: Database and Data File (Interface types: Web Service, FTP, MQ)					
Nmaew	Function	Number of Parameters	Number of Devices	Total of Parameters	ID
Flowmeter	Measure Liquid Flow Rate	2	2	4	23-24
Manometer	Measure Liquid Pressure	2	2	4	25-26
Counter	Record Switching Doors	2	4	8	27-30
Vibrator	Measure Vibration	4	10	40	31-40
	Total		40	410	

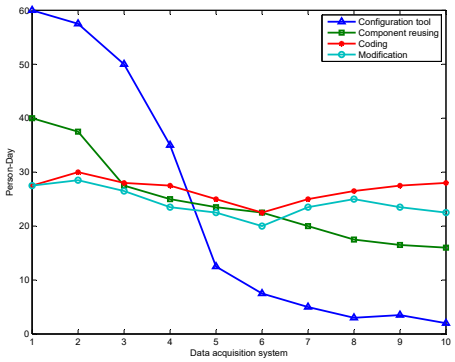


Fig. 10: Comparison of development efficiency.

TABLE VI: Information of interfaces.

Type	Count	ID
RS-232	4	1-4
RS-485	2	5-6
USB	2	7-8
Ethernet	2	9-10
Web Service	1	11
FTP	2	12-13
MQ	1	14

historical developments documents, the efficiency comparison estimated results of acquisition system development are obtained based on the content of software engineering [14]–[16]. At the beginning, it takes lots of time to develop our DAQ-Middleware and configuration tool. However, as the number of the developed systems grow, the efficiency of our method is much higher than others.

C. Data Acquisition Efficiency

DAQ-Middleware adopts the parallel data acquisition algorithm and efficiency optimization heuristic. By comparing with the serial data acquisition algorithm, we found that our methods can greatly improve the efficiency of data acquisition.

We consider a IoT system where 40 sensing devices (as shown in Table 5) are connected to the proposed middleware via 14 interfaces, which belong to 7 types as shown in Table 6. Both serial and parallel acquisition methods are deployed. The serial data acquisition algorithm sequentially obtains the data from 40 sensing devices, while the parallel data acquisition algorithm can obtain data in parallel and use the efficiency optimization heuristic to schedule the acquisition. Particularly, we sort the acquisition time of different data sources to obtain matrix G_{7*10}. According to matrix G_{7*10}, and adopting the acquisition efficiency optimization heuristic, D_{14*5} is obtained. Then perform the parallel data acquisition algorithm based on D_{14*5}, T_{14*5} also is obtained.

$$G_{7*10} = \begin{pmatrix} 19 & 19 & 16 & 15 & 0 & 0 & 0 & 0 & 0 & 0 \\ 36 & 35 & 35 & 33 & 26 & 26 & 26 & 24 & 0 & 0 \\ 57 & 56 & 56 & 55 & 55 & 0 & 0 & 0 & 0 & 0 \\ 78 & 66 & 62 & 56 & 50 & 0 & 0 & 0 & 0 & 0 \\ 80 & 51 & 50 & 46 & 45 & 45 & 0 & 0 & 0 & 0 \\ 67 & 67 & 66 & 66 & 66 & 66 & 65 & 65 & 65 & 65 \\ 52 & 51 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T_{14*5} = \begin{pmatrix} 19 & 0 & 0 & 0 & 0 & 0 \\ 19 & 0 & 0 & 0 & 0 & 0 \\ 16 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & 0 & 0 & 0 \\ 36 & 35 & 26 & 26 & 0 & 0 \\ 35 & 33 & 26 & 24 & 0 & 0 \\ 57 & 56 & 55 & 0 & 0 & 0 \\ 56 & 55 & 0 & 0 & 0 & 0 \\ 78 & 62 & 50 & 0 & 0 & 0 \\ 66 & 56 & 0 & 0 & 0 & 0 \\ \boxed{80} & 51 & 50 & 46 & 45 & \boxed{45} \\ 67 & \boxed{66} & \boxed{66} & \boxed{65} & \boxed{65} & 0 \\ 67 & 66 & 66 & 65 & 65 & 0 \\ 52 & 51 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The time required t for our algorithm to complete a data acquisition is only $t = \sum_{j=1}^{n}(t_j) = \sum_{j=1}^{n}(\max_i(T_{i,j})) = 80+66+66+65+65+45 = 387ms$, while the time required for the serial acquisition $t^{'}$ is $t^{'} = \sum_{i=1}^{m}\sum_{j=1}^{n}(T_{i,j}) = 1,979ms$. It is obviously that t is only about 1/5 of $t^{'}$. So the proposed parallel data acquisition algorithm and acquisition efficiency optimization heuristic can improve the efficiency of data

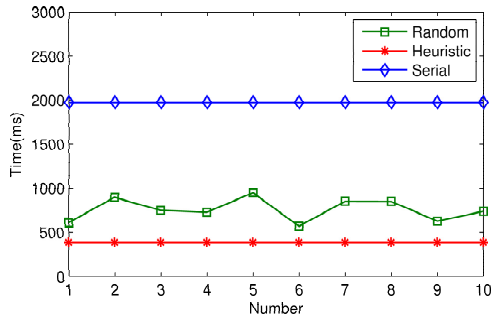


Fig. 11: Acquisition efficiency in experimental environment.

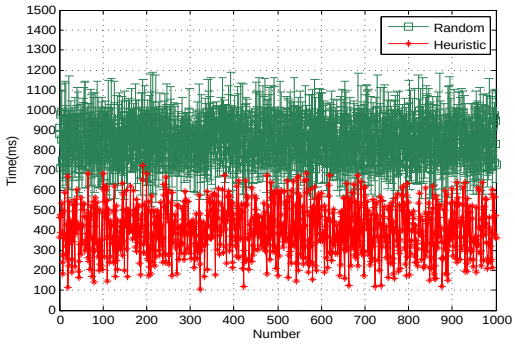


Fig. 12: Acquisition efficiency in random simulations.

acquisition. In addition, when the number of access devices and interfaces increases, such enhancment is more obvious.

At the same time, under the condition of satisfying the interface attribute constraints, we compare the proposed acquisition efficiency optimization heuristic with random allocation method in the acquisition of data sources in the experimental environment of Section 4.1. The results of the comparison of the acquisition efficiency are shown in Figure 11. The random acquisition method is based on the parallel data acquisition algorithm but use random scheduling, so its data acquisition time is lower than the serial data acquisition algorithm, but higher than the acquisition efficiency optimization heuristic.

To further validate the effect of the acquisition efficiency optimization heuristic, we simulated 1000 times. Each time we randomly generate acquisition time of 100 data sources, and compare the efficiency of the two methods (proposed method vs random allocation) in the case where the interface information has been fixed. The results of the comparison are shown in Figure 12. The acquisition time of the proposed heuristic is below the one from random acquisition method.

V. Conclusion

This paper has put forward a data acquisition middleware design based on IoT (DAQ-Middleware), which can realize fast and flexible access to heterogeneous data sources in different fields. DAQ-Middleware is located between data sources and higher-level IoT application systems, making them more transparent from each other. Through the definition of the data source description file, the DAQ-Middleware structure model and the standardized interface interactive content, we only need to modify the data source description file to achieve data sources addition, modification and deletion. At the same time, based on the characteristics of DAQ-Middleware structure model, we also propose a parallel data acquisition algorithm and acquisition efficiency optimization heuristic to reduce data acquisition time. Experimental comparisons with the traditional data acquisition algorithm confirm that our methods can improve the efficiency of data acquisition. Taking into account a very wide range of factors in the usage of IoT applications, in the future, we will verify, compare and analyze DAQ-Middleware's performance in more areas of IoT applications.

Acknowledgment: Z. Qiu and S. Guo are supported by the fellowship from the China Scholarship Council (CSC) under Nos. 201606330018 and 201606330021. This work is also partially supported by the National Natural Science Foundation of China under Nos. 61379127, 61572448 and 61572347.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] M. Peng, Y. Li, Z. Zhao, and C. Wang, "System architecture and key technologies for 5G heterogeneous cloud radio access networks," *IEEE network*, vol. 29, no. 2, pp. 6–14, 2015.

[3] T. Riedel, N. Fantana, A. Genaid, D. Yordanov, H. R. Schmidtke, and M. Beigl, "Using web service gateways and code generation for sustainable IoT system development," in *Proc. of Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[4] H. Ning and Z. Wang, "Future internet of things architecture: like mankind neural system or social organization framework?" *IEEE Communications Letters*, vol. 15, no. 4, pp. 461–463, 2011.

[5] M. Gigli and S. Koo, "Internet of things: services and applications categorization," *Advances in Internet of Things*, vol.1, no.2, p. 27, 2011.

[6] Z. Qiu, Z. Guo, S. Guo, L. Qiu, X. Wang, S. Liu, and C. Liu, "IoTI: Internet of things instruments reconstruction model design," in *Proceedings of 2016 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, 2016, pp. 1–6.

[7] S. D. T. Kelly, N. K. Suryadevara, and S. C. Mukhopadhyay, "Towards the implementation of IoT for environmental condition monitoring in homes," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3846–3853, 2013.

[8] Z. Qiu, N. Hu, Z. Guo, L. Qiu, S. Guo, and X. Wang, "IoT sensing parameters adaptive matching algorithm," in *Proc. of International Conf. on Big Data Computing and Communications*, 2016, pp. 198–211.

[9] V. S.-K. Kovac, "Virtual instrumentation and distributed measurement systems," *J. Electr. Eng*, vol. 55, no. 1-2, pp. 50–56, 2004.

[10] Z. Qiu, Z. Guo, and C. Liu, "Adaptive high-speed data acquisition algorithm in sensor network nodes," *Journal of Southeast University. Natural Science Edition*, vol. 42, 2012.

[11] Z. Guo, C. Liu, X. Wang, H. Ma, Y. Jiang, B. Zheng, and B. He, "CVIS: Complex virtual instruments system architecture," in *Proc. of 2013 IEEE International Instrumentation and Measurement Technology Conf. (I2MTC)*. IEEE, 2013, pp. 566–571.

[12] K. Hu, Z. Guo, et al., "Composition model of complex virtual instrument for ocean observing." *JSW*, vol.9, no.5, pp.1177–1188, 2014.

[13] D. M. Toma, T. O'Reilly, J. del Rio, K. Headley, A. Manuel, A. Bröring, and D. Edgington, "Smart sensors for interoperable smart ocean environment," in *Proc. of OCEANS, 2011 IEEE-Spain*. IEEE, 2011, pp. 1–4.

[14] R. L. Jones and A. Rastogi, "Secure coding: building security into the software development life cycle," *Information Systems Security*, vol. 13, no. 5, pp. 29–39, 2004.

[15] A. Ramaswami, T. Hillman, B. Janson, M. Reiner, and G. Thomas, "A demand-centered, hybrid life-cycle methodology for city-scale greenhouse gas inventories," *Environmental Science & Technology*, vol. 42, no. 17, pp. 6455–6461, 2008.

[16] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches: A survey," *Annals of software engineering*, vol. 10, no. 1-4, pp. 177–205, 2000.