A Data Stream Cleaning System Using Edge Intelligence for Smart City Industrial Environments

Danfeng Sun, *Member, IEEE*, Shan Xue, Huifeng Wu, *Member, IEEE*, and Jia Wu, *Senior Member, IEEE*

***Abstract*—Cities are becoming smarter because of recent advances in artificial intelligence and the Internet of Things. However, heterogeneous data source in smart cities are continuously producing low-quality data, and ever-growing applications have greater real-time requirements. Therefore, this article proposes a data stream cleaning system (named DSCS) using edge intelligence to utilize the advantages of cloud servers and edge devices. The DSCS in edge nodes consists of a dynamic protocol interpreter, a structure parser, and a cleaning model activator. Meanwhile, a cloud server, which has pools of protocol and structured programs and cleaning models, supports the edge nodes to adapt massive heterogeneous data sources. To validate the proposed data cleaning system, we applied it to two scenarios: monitoring the injection molding machines, and base stations. The DSCS can have a stable processing time when the number of accessed edge devices is increased, as well as a good cleaning effect.**

***Index Terms*—Artificial intelligence (AI), data stream, edge computing, Industrial Internet of Things (IoT).**

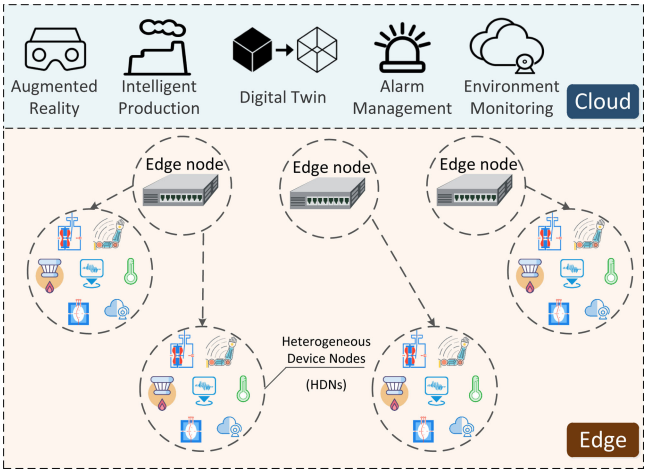

Fig. 1. Typical dataflow scenario in a smart city.

I. Introduction

EMERGING technologies in fields, such as artificial intelligence (AI) and the Internet of Things (IoT), have attracted much interest in making cities smarter. Comprehensively taking these technologies into consideration, data should be the key of smart cities. In general, massive heterogeneous data sources in a smart city produce data continuously, and these data are mainly harvested by data acquisition systems (DASs). Fig. 1 depicts a typical data flow scenario in a smart city. The data sources, such as sensors, devices, and vehicles, connect with different kinds of cloud servers through edge nodes, which use wired or wireless (3G/4G/5G, WiFi, Bluetooth, Zigbee) networks to communicate with data sources and cloud servers. Because of the heterogeneity of the data sources in a smart city, many customized DASs have been implemented. These DASs always meet the requirement of low-quality data cleaning from heterogeneous data sources. The data cleaning process generally has three stages: data interpretation, data structuring, and data cleaning.

Data interpretation leverages specific communication protocols (e.g., OPC/UA, Modbus/TCP) to collect specific data from data sources. In the context of the IoT, multisource heterogeneity of data sources has been hindering data interpretation for a long time. Reconfiguration methods have been widely explored to alleviate this problem with different types of hardware, such as field programmable gate arrays (FPGAs) and programmable logic controllers (PLCs). PLC technology has been used to implement reconfigurable interfaces [1]. Notably, Bao *et al.* [2] devised a data acquisition method for FPGAs that combines dynamic and static system reconfiguration. Wu *et al.* [3] provided a dynamic configuration method supported by cloud servers

Manuscript received August 2, 2020; revised October 18, 2020, December 26, 2020, and April 18, 2021; accepted May 1, 2021. Date of publication May 6, 2021; date of current version October 27, 2021. This work was supported in part by the National Key R&D Program of China under Grant 2019YFB1705102 and in part by the Science and Technology Program of Zhejiang Province under Grant 2021C01187. Paper no. TII-20-3764. *(Corresponding author: Huifeng Wu.)*

Danfeng Sun is with the Institute of Industrial Internet, Hangzhou Dianzi University, Hangzhou 310018, China (e-mail: df@hdu.edu.cn).

Shan Xue is with the Department of Computing, Macquarie University, Sydney, NSW 2109, Australia, and also with CSIRO's Data61, Sydney, NSW 2121, Australia (e-mail: emma.xue@data61.csiro.au).

Huifeng Wu is with the Institute of Intelligent and Software Technology, Hangzhou Dianzi University, Hangzhou 310018, China (e-mail: whf@hdu.edu.cn).

Jia Wu is with the Department of Computing, Macquarie University, Sydney, NSW 2109, Australia (e-mail: jia.wu@mq.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2021.3077865.

Digital Object Identifier 10.1109/TII.2021.3077865

to minimize the protocol programs in edge nodes. In addition, commercial companies enrich their own hardware by increasing resources or enhancing performance to address the data interpretation problem, e.g., Omron's CP1W expansion units [4].

Data structuring involves arranging collected data into a specific structure. In frequent pattern mining, some tree structures are applied to extract uncertain frequent data [5], [6]. In this process, the given uncertain data are loaded into the main memory in their own tree forms. In this case, the tree may be too large for the available memory. Lee and Yun [7] proposed list-based data structures to ensure efficient data mining without pattern losses. Some structures can be repeated in one-time data processing and, hence, the chunks of a piece of structured data should be considered. To address this, Haque *et al.* [8] proposed an efficient semisupervised framework that can determine chunk boundaries dynamically, and Verma and Kumar [9] presented adaptive data chunk scheduling for concurrent multipath transfer.

Data cleaning is considered for most applications because of the low quality of the raw data. This process mainly includes data filtering, filling, and repair. In practice, most data have time and space associations, and using these associations can improve the speed and accuracy of data filtering. Garofalakis *et al.* [10] applied this principle in sensor data filtering. Employing probability and statistics theory to find abnormal data in high-dimensional space or eliminate redundant data is another hot research topic; for example, Maletic and Marcus [11] pointed out that if most data recording follows a rule and some data do not obey this rule, then these data are abnormal and can be filtered. Fuzzy matching is another data filtering method. Ma *et al.* [12] used a similarity function to calculate the similarity between data, and then filtered repeated data. Massive datasets can have missing data, which hinders data analysis. Hence, Galárraga *et al.* [13] proposed an association rule to fill in missing data. Kong *et al.* [14] presented a space–time constrained, compressed sensing algorithm to reconstruct lost data. Xiang *et al.* [15] presented a multisource learning method to blockwise fill missing data. In contrast to data filling, data repair focuses on changing the existing data. Data can be repaired by experts, statistical methods, and machine learning using cost functions to minimize the distance between the original database and modified database [16]. For example, Hao *et al.* [17] presented a novel cost-based model to quantify the cost of data repair, thereby improving the performance of both detecting and repairing errors; and Ye *et al.* [18] proposed a multisource data repair method under integrity constraints and source reliability.

The aforementioned three stages always appear together in massive types of applications, however, the research of uniform and flexible data cleaning systems is lagging compared with the separate stage research because of the multisource heterogeneity of data sources, and the complexity of structuring these heterogeneous data for many uncertain cleaning requirements. Furthermore, recent advances in AI are improving the effect of cleaning models, but also challenging the computing pattern. Past cloud computing is suffering from high response latency and high cost of data transmission [19], [20], and edge computing only has limited resources [21]. Resource offloading between cloud servers and edge nodes can take into account the advantages of both cloud and edge computing [22]–[24]. Baccarelli *et al.* [22] made a survey and pointed out the two aspects of the consideration of resource offloading in big data stream mobile computing: energy efficiency and real-time processing. They conducted a detailed case named StreamCloud to optimize energy consumption. Since the stream data cleaning is very relevant to its inner time and space relationships, the real-time feature using edge intelligence is one key point of our article.

Thus, we propose a data stream cleaning system (DSCS) leveraging edge intelligence. The main contributions of this article are as follows.

1) We propose a uniform and flexible DSCS with three layers to adapt variable application requirements. The three layers are a dynamic protocol interpreter for collecting data from data sources, an structure parser for structuring data, and a cleaning model activator for cleaning data.
2) Edge intelligence is applied in the DSCS where the edge devices run intelligent reasoning models optimized by the cloud servers.
3) To validate the proposed DSCS, we applied it in two scenarios, namely, monitoring systems of base stations and injection molding machines (IMMs).

The rest of this article is organized as follows. Section II introduces the architecture and details of the DSCS. Two experiments are conducted in Sections III. Finally, Section IV concludes this article.

II. Data Stream Cleaning System

The DSCS is based on a cloud-edge structure, where cloud servers are responsible for optimizing the cleaning models in edge devices, and edge devices execute the data stream cleaning process. The DSCS has three parts: a dynamic protocol interpreter, a structure parser, and a cleaning model activator. Fig. 2 shows the data processing flow of the DSCS. The dynamic protocol interpreter recognizes data in specified protocols from different data sources and transfers the data to the structure parser. The structure parser structures data as multidimensional input for adaptive cleaning models (ACMs). Then, the cleaning model activator reacts these models to clean the low-quality data, and upload these data to the cloud server or store them in a local database.

Fig. 3 shows these three parts in cloud servers and edge devices. There can be many customized cleaning models and structured and protocol programs in cloud servers. Only the required ones are downloaded into edge devices according to configuration tables (CTs). In the edge devices, these models and programs are organized by the dynamic protocol interpreter, structure parser, and cleaning model activator in turn to produce high-quality data for upper-layer applications.

The three parts are explained in detail as follows.

A. Dynamic Protocol Interpreter

The protocol interpreter in the DSCS is similar to the dynamic edge access system in [3]. In that system, the protocol programs can be dynamically downloaded and updated, and are supported

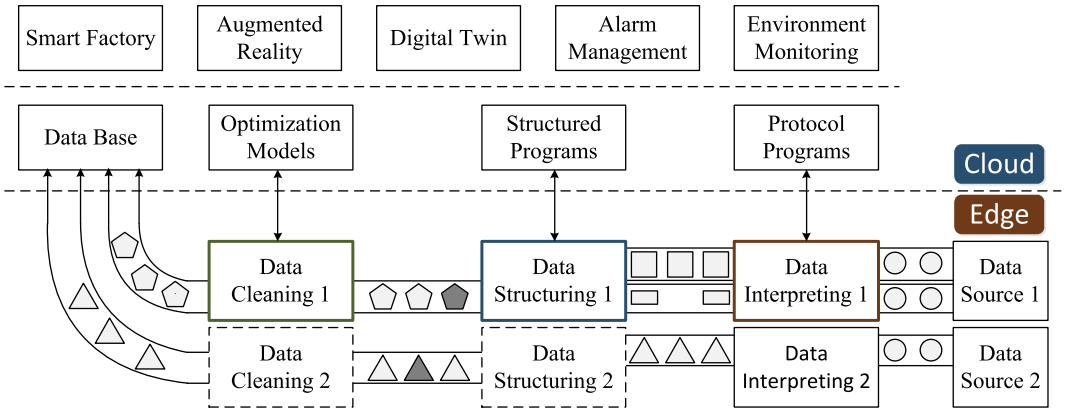

Fig. 2. Overview of the DSCS. The data produced by the data sources go through data interpretation, structuring, and cleaning, in turn.

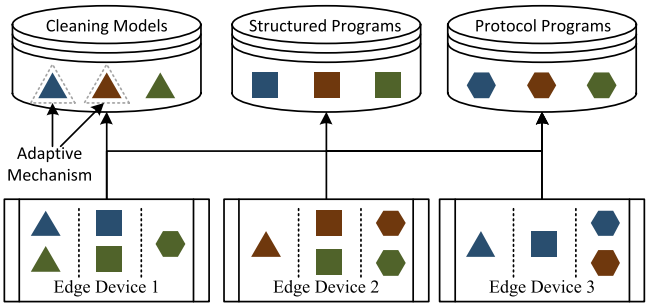

Fig. 3. Cloud server with cleaning models, structured programs, and protocol programs supporting the edge devices. The cleaning models can be adaptive.

by a CT and corresponding device templates (MTs). The CT records the information of the system access devices. The MT uniquely corresponds to an access device, and stores the information of all required collected parameters, which are produced in terms of the CT. Table I shows an MT example of an IMM. In particular, the SID denotes which structured program is used to structure this parameter's value after data interpretation.

Experts can design CTs for different scenarios. The cloud server can then produce MTs combined with the CT and download them to edge devices. Algorithm 1 is used to interpret protocols and harvest data from data sources. It reads every item in the CT in turn, and then reads the MT with its MID. Then, it traverses the MT and gets every row. If the collecting interval is reached, and the address of this parameter is not null, and the algorithm reads data values from data sources with the specific address and protocol. Then, the algorithm creates a list to save the SIDs of MT; this list is called *lst*. If an MT finishes, its acquisition flag (fA[cMID]) recovers to 0. If the protocol interpreter ends, acquisition and structuring states are set to 0 and 1, respectively.

Algorithm 1: Protocol Interpreter.

```
Input: CT
Output: lst
foreach itemC in CT do
    fA[itemC.MID] = 1;
    cMID = itemC.MID;
    foreach itemM in itemC.MT do
        if time > lastTimetamp[itemM.PID]+itemM.T1
         then
            fS[itemM.SID] = 1;
            address = itemM.address;
            if address != null then
                R = readData(address,itemC.protocol);
                lastTimetamp[itemM.PID] = time;
            end
        end
        foreach SID in itemM.SID do
            if !exist(SID) then
                addS(lst,SID);
            end
            if !exist(SID,itemM.PID) then
                addP(lst,SID,itemM.PID);
            end
            lst[itemM.SID][itemM.PID] = R;
        end
    end
    fA[cMID] = 0;
end
acqD = 0;
strD = 1;
```

B. Structure Parser

Structure parser is responsible for structuring rough data into an input data for a specific cleaning model. The process of structure parser is abstracted as Algorithm 2. It traverses unstructured data that is stored in *lst*. If there is a "c" in the parameter's type, then the algorithm deals with its equation to get its value, which means the value is calculated from other associated parameters. It contains six steps to get the parameter value (V), which are as follows.

1) Getting equation (E_o) string from MT, which has parameters names.
2) Extracting parameters names (pNames) from E_o.

TABLE I
MACHINE TEMPLATE CASE

PID	Parameter Name	Group	Unit	Property	Type	Address	Equation	T1	SID
0x3EE	sv_dLastCycleTime	Overview	s	r/c	LINT	SVs.system.sv_dLastCycleTime	sv_dLastCycleTime/1000000	0X3E8	0X00
0x3F6	sv_iShotCounterRetain	Overview	-	r	DINT	SVs.system.sv_iShotCounterRetain	-	0X3E8	0X00
0x3FE	sv_OperationMode	Overview	-	r	DINT	SVs.system.sv_OperationMode	-	0X3E8	0X00
0x3FF	sv_bHeatingOn	Overview	-	r	BOOL	SVs.HeatingNozzle1.sv_bHeatingOn	-	0X3E8	0X00
0x3ED	sv_bMotorStarted	Overview	-	r	BOOL	SVs.Motor1.sv_bMotorStarted	-	0X3E8	0X00
0x2712	sv_ti_OilTemp	OtherData	°C	r/w	REAL	SVs.OilMaintenance1.ti_OilTemp	-	0X3E8	0X00
0x2718	sv_rCushion	OtherData	mm	r/w	REAL	SVs.Injection1.sv_rCushion	-	0X3E8	0X00
0x2719	sv_rCutOffPosition	OtherData	mm	r/w	REAL	SVs.Injection1.sv_rCutOffPosition	-	0X3E8	0X00
0x271A	sv_rCutOffPressure	OtherData	bar	r/w	REAL	SVs.Injection1.sv_rCutOffPressure	-	0X3E8	0X00
0x271B	sv_rPlastEndPosition	OtherData	mm	r/w	REAL	SVs.Injection1.sv_rPlastEndPosition	-	0X3E8	0X00
0x2AFE	sv_rMaxSpeedInject	OtherData	mm/s	r/w	REAL	SVs.Injection1.sv_rMaxSpeedFwd	-	0X3E8	0X00

3) Mapping the pNames to parameters IDs (pIDs).
4) Reading parameters values (pValues) from lst according to pIDs.
5) Replacing pNames in E_o with pValues to obtain an equation in numerical form (E_v).
6) Executing the equation to get the V.

Note that only some parameters have an equation. Then, it calls a specific structured program in terms of cSID. Note that if the SID is 0, it is a default option for most data structuring cases, which only parameters' equations are dealt with. If any structured program is end, its flag (fS[cSID]) and relevant cleaning model flag (fC[cCMID]) are set 0 and 1, respectively, where cSID and cCMID are current structured program and cleaning model IDs.

Algorithm 2: Structured Parser.

```
Input: lst, CMT, MT
Output: lst
foreach itemS in lst do
    if !fS[itemS.SID] then
        continue;
    end
    cSID = itemS.SID;
    cCMID = getCMID(cSID,CMT);
    foreach itemP in itemS do
        type = getType(itemP.PID, MT);
        if type.has('c') then
            E_o = getE(itemP.PID, MT);
            pNames = getPName(E_o);
            pIDs = getPID(MT,pName);
            pValues = readVal(lst,pIDs);
            E_v = replace(E_o,pNames,pValues);
            V = exeEqu(E_v);
            lst[cSID][itemM.PID] = V;
        end
    end
    if cSID != 0 then
        callSP(cSID);
    end
    fS[cSID] = 0;
    fC[cCMID] = 1;
end
strD = 0;
clnD = 1;
```

C. Cleaning Model Activator

The cleaning model activator works as abstracted in Algorithm 3. It traverses cleaning models included in a cleaning model table (CMT) to check whether any cleaning model flag is 1. If this is the case, the algorithm starts the cleaning model. The algorithm compares the obtained chunks from the corresponding structured program with the total required chunks. If they match, the algorithm inputs the chunks into the cleaning model, obtains the output, and recovers fC[cCMID] to 0. If all cleaning models finish cleaning tasks, *clnD* and *acqD* are set to 0 and 1, respectively. In particular, if cCMID is 0, no cleaning model works on the chunks. Note that the aforementioned CMT is a table recording the CMID and its related *model name*, *SID*, *type*, *T2*, and *chunks*. Here, if *type* equals 0, the data cleaning works with RAM and directly uploads to cloud servers after cleaning; if *type* equals 1, the data cleaning works with a local database and directly uploads to cloud servers; and if *type* equals 2, the cleaned data are stored in a local database, and only when cloud servers request the data are the data uploaded. An example is shown in Table II.

To cope with the changing monitoring environment, the cleaning models can be adaptively optimized. Fig. 4 shows the execution mechanism of an ACM. Cloud servers collaborate with edge devices. The cloud servers are responsible for online training and reasoning. Through a decision mechanism, the optimized reasoning model can be updated into edge devices. The decision mechanism varies across different applications. For example, it can be used to calculate the mean square error between online and offline reasoning results. In typical scenarios, the edge devices have two types of nodes: training nodes and reasoning nodes. The training nodes provide training data for training or optimizing cleaning models, and reasoning nodes are applied targets. Note that the training and reasoning nodes should have the same classification. In general, we assume that the nodes

TABLE II
EXAMPLE CMT

CMID	Model Name	OID	SID	Type	T2	Chunks
0x00	Default	0x00	0x00	2	0	1
0x01	Upper-Lower Bound	0x01	0x01	1	0	1
0x02	LSTM-based Noise Removing	0x02	0x02	0	0	12

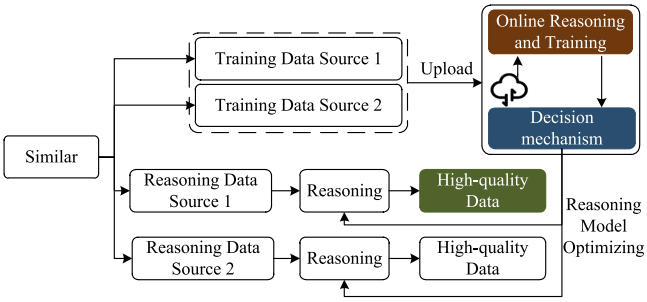

Fig. 4. Process of cleaning models with an adaptive feature.

Algorithm 3: Cleaning Model Activator.

```
Input: lst, CMT
Output: output
foreach itemCM in CMT do
    if !fC[itemCM.CMID] then
        continue;
    end
    cCMID = itemCM.CMID;
    cSID = getSID(cCMID,CMT);
    chunk[cCMID]++;
    ck=itemCM.chunk;
    lstCM[cCMID].add(lst[cSID]);
    if chunk[cCMID] == ck then
        if cCMID != 0 then
            output = callCM(cCMID,lstCM[cCMID]);
        end
        else
            output = lstCM[cCMID];
        end
        lstCM[cCMID].remove();
        chunk[cCMID] = 0;
        fC[cCMID] = 0;
    end
end
lst.remove();
clnD = 0;
acqD = 1;
```

sharing the same MTs for the same cleaning models have the same classification.

D. FSM for DSCS

In this article, an FSM is designed for \precisely debugging the DSCS. The FSM is assumed to be a five tuple [25]

$$M = (S, X, Y, \delta, \lambda) \tag{1}$$

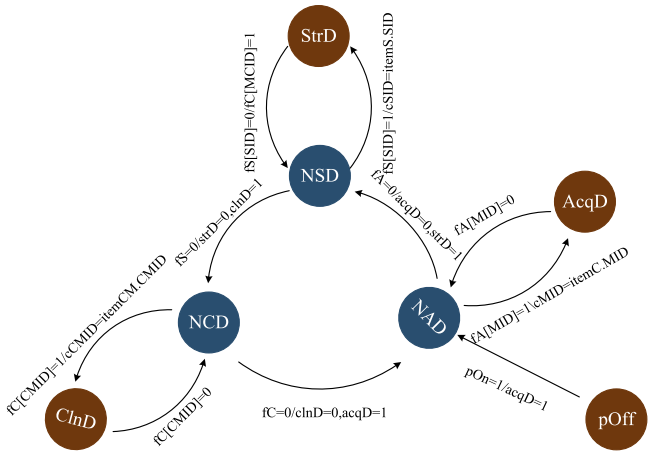

Fig. 5. Finite state machine of the DSCS.

where $S = \{s_1, \ldots, s_l\}$ is the finite set of states. $X = \{x_1, \ldots, x_p\}$ and $Y = \{y_1, \ldots, y_q\}$ are finite sets of input and output, respectively. δ and λ are state transition and output functions. Note that $\lambda : S \times X \to S$, and $\lambda : S \times X \to Y$. If an edge device is in state s and input x comes, then the edge device transfers to state s' (i.e., $s' = \delta(s, x)$) and outputs y (i.e., $y = \lambda(q, x)$), and this whole process denotes $\tau = (s, x/y, s')$.

In our specific case, M is depicted in Fig. 5. The set S includes $pOff, AcqD, StrD, ClnD, NAD, NSD$, and NCD, where $pOff$ is the state before edge devices power ON; $AcqD$, $StrD$, and $ClnD$ are state sets of acquiring, structuring, and cleaning data, respectively; and NAD, NSD, and NCD are states of preparing to switch to the next acquisition program, structured program, and cleaning model. Regarding the set X, $p\text{On} = 1$ is an input when edge devices power ON. $fA[\text{MID}] = 0$ or 1, $fS[\text{SID}] = 0$ or 1, and $fC[\text{CMID}] = 0$ or 1 are inputs about checking flags of acquiring, structuring, and cleaning data whether their values are 0 or 1. Note that these flags without indices denote flags with any index meet conditions; for example, $fA = 0$ denotes none $fA[\text{MID}]$ equals 1. The set Y contains changes of states and the current ID assignment. For example, $acqD = 0$ and $strD = 1$ are state changes of $acqD$ and $strD$ to 0 and 1, respectively; and $c\text{MID} = \text{item}C.\text{MID}$ denotes the value of item$C$.MID, which is assigned to the current MID cMID.

There are two types of state transition process τ: outer transitions and inner transitions. The outer transition τ_o is a state transition between different data processing stages; for example, $\tau_o^1 = \{pOFF, pON = 1/acqD = 1, NAD\}$ denotes that the state transfers from $pOFF$ to NAD. The inner transition τ_i is a state transition within one stage, e.g., $\tau_i^1 = \{NAD, fA[\text{MID}] = 1/c\text{MID} = \text{item}C.\text{MID}, AcqD\}$ denotes a state transfer from NAD to $AcqD$ in the data acquisition stage.

The set τ_o is as follows:

$$\begin{cases} \tau_o^1 = \{pOFF, pON = 1/acqD = 1, NAD\} \\ \tau_o^2 = \{NAD, fA = 0/(acqD = 0, strD = 1), NSD\} \\ \tau_o^3 = \{NSD, fS = 0/(strD = 0, clnD = 1), NCD\} \\ \tau_o^4 = \{NCD, fC = 0/(clnD = 0, acqD = 1), NAD\} \end{cases}. \tag{2}$$

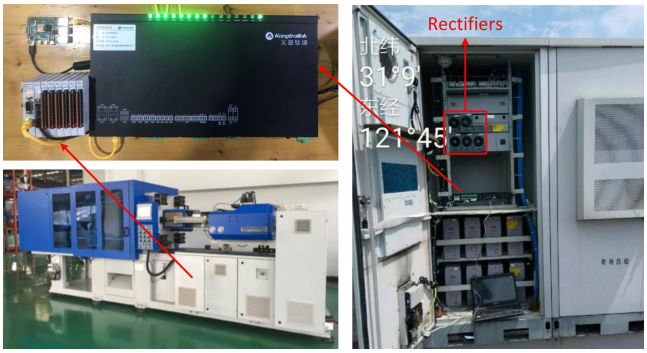

Fig. 6. Testbed consists of a Raspberry Pi, an IMM controller, and an FSU.

The set τ_i is as follows:

$$\begin{cases} \tau_i^1 = \{NAD, fA[\text{MID}] = 1/c\text{MID} = \text{item}C.\text{MID}, AcqD\} \\ \tau_i^2 = \{AcqD, fA[\text{MID}] = 0/\text{null}, NAD\} \\ \tau_i^3 = \{NSD, fS[\text{SID}] = 1/c\text{SID} = \text{item}S.\text{SID}, StrD\} \\ \tau_i^4 = \{StrD, fS[\text{SID}] = 0/\text{null}, NSD\} \\ \tau_i^5 = \{NCD, fC[\text{CMID}] = 1/c\text{CMID} = \text{item}CM.\text{CMID}, ClnD\} \\ \tau_i^6 = \{ClnD, fC[\text{CMID}] = 0/\text{null}, NCD\} \end{cases} \tag{3}$$

III. Experiment

The DSCS is implemented in edge devices and supported by cloud servers. In our testbed, the edge devices adopt Raspberry Pi 4B with 1.5-GHz 4-core CPU and 4-GB RAM, and the cloud server uses Ali Cloud with 2.5-GHz 2-core CPU, 8-GB RAM, 40-GB hard disk, and 1–4 M bandwidth. Fig. 6 shows our testbed: a Raspberry Pi 4B, an IMM control system, and a field supervision unit (FSU). The IMM control system is a distributed system containing a CPU module, a power module, digital input and output modules, analog input and output modules, and temperature modules; this system is used in Scenario 1. The FSU is used to monitor base stations, and is used in Scenario 2. To indicate the performance of data cleaning, we define the time of obtaining high-quality data as follows.

Definition 1 (Time of obtaining high-quality data $\mathcal{H}$): If t_1 is the moment that an edge device starts collecting data from data sources, and t_2 is the moment when data cleaning is completed, then $\mathcal{H} = t_2 - t_1$.

In contrast to the testbed in Fig. 6, we used a Lenovo ThinkCentre with 2.11-GHz 4-core CPU and 12-GM RAM to simulate the IoT environment for cleaning data. In this desktop computer, we randomly create threads between 0 and 5 s, and run the same program with the testbed to simulate that many Pis are working with the testing system, i.e., a thread simulates the work of a Pi. Then, when we change the number of threads, and the influence of other Pis on our testbed can be known.

A. Scenario 1: Processing Data From IMMs

In the production of IMMs, monitoring production parameters is important to avoid defective products. In general, there are two typical options: monitoring five production parameters or monitoring 11 production parameters. These are denoted as p_5 and p_{11}, respectively, and are listed in Table I. Note that parameters of p_5 are previous five ones.

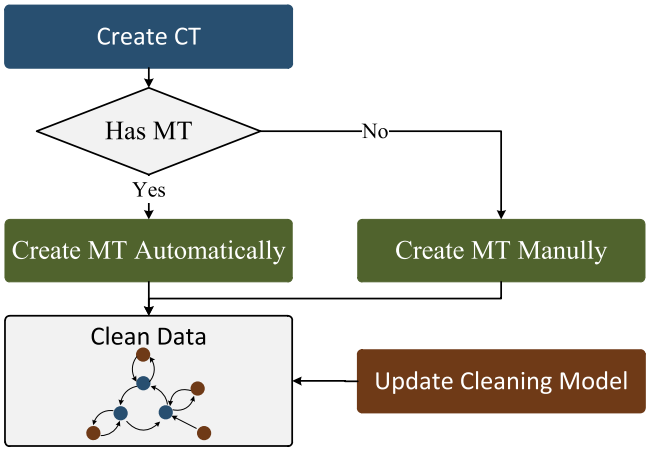

Fig. 7. Process of the DSCS.

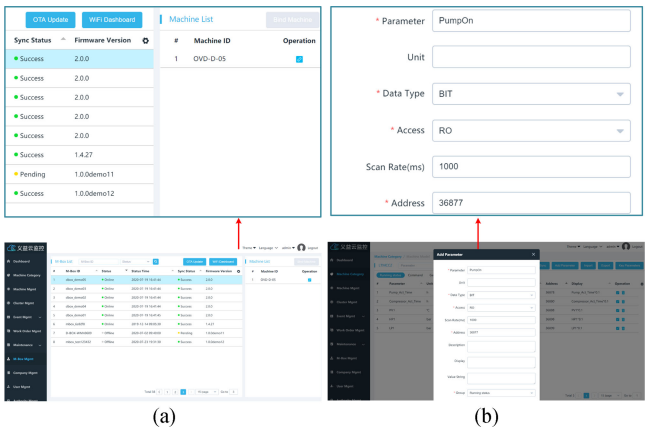

Fig. 8. (a) Part of a CT in the IMM management system. (b) Demonstration of how to add one parameter in one MT.

The DSCS is merged into an IMM edge-cloud management system in Scenario 1. Fig. 7 shows the process of the DSCS in the IMM management system. In the cloud server, the CT is created like a realistic case. MTs are produced automatically or manually. Note that the system generally provides most MTs. Then, the system downloads the initial protocol programs, structured programs, and cleaning models into the accessed IMMs according to the MT. Thus, the DSCS starts to clean data. Fig. 8(a) is a part of a CT in the IMM management system, and Fig. 8(b) shows how to add one parameter in one MT. After one IMM accepts customized programs from the cloud server, the Pi hot starts and makes transition τ_o^1, and the system changes state from pOff to NAD. Then, the DSCS collects data (i.e., parameters of p_5 or $p_1 1$) in terms of the MT. In this stage, the state switches between NAD and $AcqD$ when the system gets every parameter's value from the IMM. When all parameters are collected, transition τ_i^2 reacts, and the state changes to NSD. The data are structured into required

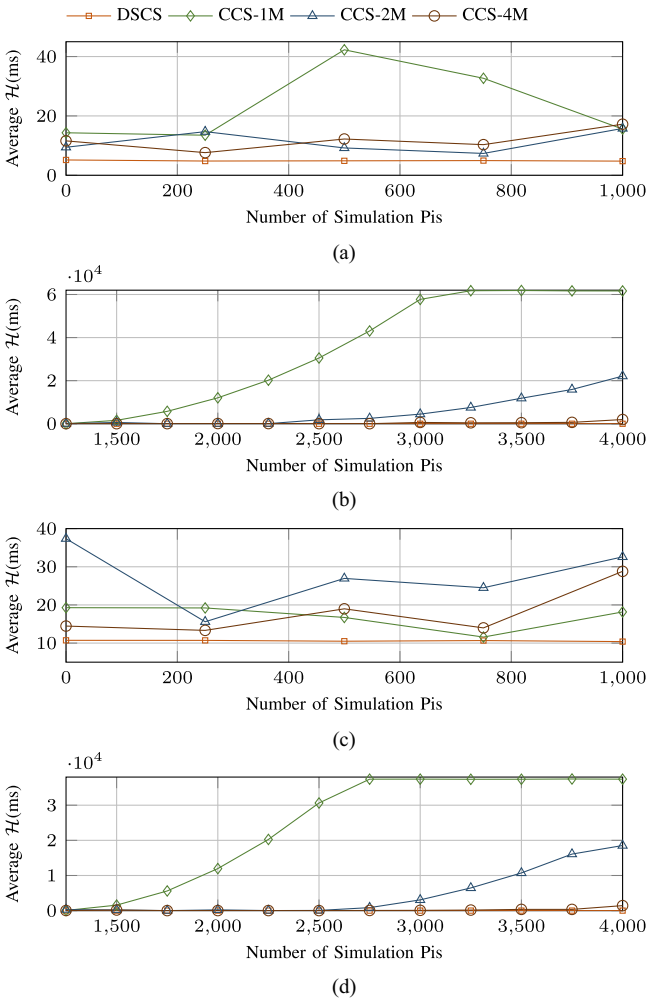

Fig. 9. Two cases of average $\mathcal{H}$ against the number of Pis in the DSCS and CCS, which are divided into two parts. (a) p_5 within 1000 Pis. (b) p_5 over 1250 Pis. (c) p_{11} within 1000 Pis. (d) p_{11} over 1250 Pis.

forms with the state changing between NSD and $StrD$. If all structured programs finish, transition τ_i^3 reacts, and the state shifts to NCD. Then, the structured data can be cleaned by corresponding cleaning models, and the state changes between $ClnD$ and NCD accordingly. When all data are cleaned, the state changes back to NAD with the transition τ_i^4.

In this experiment, we mainly compared the DSCS with a cloud-based cleaning system (CCS) from the aspect of $\mathcal{H}$. The CCS also used the Raspberry Pi collecting data but cleaning data in the cloud. In addition, the CCS adopted a single thread to deal with the edge requests. Because this experiment has no relationship with the meaning of the parameter values, we collected data directly from the control system, which was not installed in an IMM. The MT of p_{11} is shown in Table I. Structured methods of these parameters are default methods. p_5 is similar to p_{11}. To test the influence of bandwidth in CCS, we varied it from 1 to 4 M in every experiment, and we distinguish them with additional symbols in the legend. For example, CCS-1 M denotes using 1-M bandwidth in the CCS.

Fig. 9 shows $\mathcal{H}$ versus the number of Pis under the situations of p_5 and p_{11}. We increased the number of Pis from 0 to 4000 with an interval of 250, and tested $\mathcal{H}$ with the DSCS and CCS. Note that $\mathcal{H}$ is an average value of ten-time $\mathcal{H}$ in Fig. 9. In Fig. 9(a), $\mathcal{H}$ of the DSCS remains around 4.9 ms, which is shorter than the CCS's. However, in Fig. 9(b), $\mathcal{H}$ of the CCS makes a sharp change to around 12.0, 11.8, and 2.0 s, respectively, when the number of Pis increased accordingly to 2000, 3500, and 4000. This sharp change occurred because the cloud server could not process the requests immediately, and the requests were put into a waiting queue. Finally, $\mathcal{H}$ of the CCS stays at around 60, 22, and 2 s, because of the packet loss in the cloud server. As shown in Fig. 9(c), $\mathcal{H}$ of the DSCS can still remain at around 10 ms, because the DSCS processed the data in every single Pi, which was not influenced by others. Again, $\mathcal{H}$ of the CCS became worse when the number of Pis increased in Fig. 9(d). We can see that the bandwidth has a little influence on $\mathcal{H}$ when the system accesses a few Pis, e.g., less than 1000, while low bandwidth is faster falling into the issue of the request block.

B. Scenario 2: Processing Data From Base Stations

To validate the adaptive characteristic of cleaning models, an experiment on base station monitoring was conducted. In the FSU, we saved two real datasets collected from another FSU located in Pudong New Area, Shanghai, China, where latitude and longitude are 31°9' and 121°45'. The data were harvested from December 29, 2018 to January 31, 2019. In that base station, there are four rectifiers monitored by the FSU. We chose three of them as the training data source, and used the remaining one as the reasoning data source. The training and reasoning data sources provided 5500 and 1375 data items, respectively, and all data had three relevant dimensions: ambient temperature, rectifier current, and rectifier temperature. Here, we applied a cleaning model for filtering noise points about rectifier temperature. 10% of the data provided by the reasoning data source were given an additional normal distribution noise between 1.0 and 1.2 °C on the rectifier temperature. Then, a Raspberry Pi 4B installed with the DSCS was used to collect, structure, and clean the data from the FSU. The data flow is as shown in Fig. 4. The training and reasoning data were collected in parallel. The initial cleaning model was downloaded to the Raspberry Pi 4B after training using 2200 data. Then, whenever an additional 1100 points were used to train an optimized model, the model was updated with the reasoning model in the Raspberry Pi 4B to improve the filtering performance.

The structured method herein only combined the latest three-dimension data with its previous 11 time-series data and used this as input into a long short-term memory-based (LSTM-based) cleaning model. LSTM is a kind of recurrent neural network and has been widely applied [26].

Table III shows the neural network structure of the LSTM-based cleaning model, which has an LSTM layer and a fully connected layer (FNN); they contain 64 and 32 neurons, respectively. The activation function of the FNN is a rectified linear unit (ReLU). The optimizer is *adam*. The loss function is the

TABLE III
NEURAL NETWORK STRUCTURE OF THE LSTM-BASED CLEANING MODEL

Input	LSTM	FNN	Output	Opt	Loss	LR	Epoch	BS
3*12	64 tanh	32 relu	1	adam	MAE	0.001	800	500

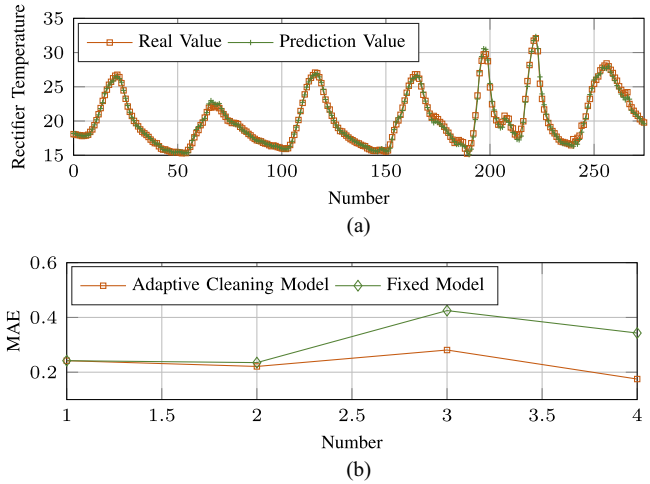

Fig. 10. (a) Prediction result of initial model. (b) Comparison of the MAE of the ACM with a fixed model, which is the same with the initial one of the ACM.

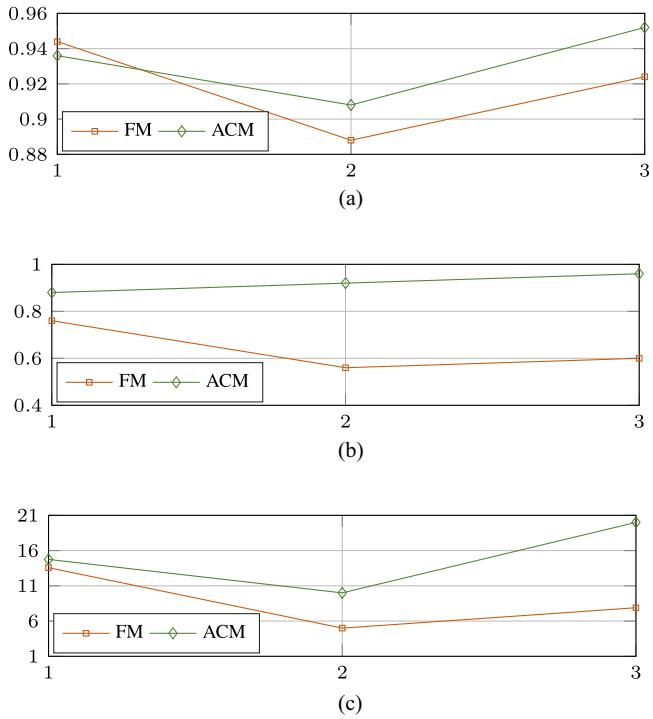

Fig. 11. Three ratios versus the number of processing data to evaluate the cleaning effect. (a) Falsely filtered ratio. (b) Noise filtered ratio. (c) Filtered ratio.

mean absolute error (MAE). The learning rate (LR) is 0.001. The epochs and batch size (BS) are 800 and 500, respectively.

The cleaning model outputs a prediction value, and then adds and subtracts a threshold on this prediction value produced a pair, namely upper and lower bounds, denoted as u and l. Note that the threshold was set to 1 °C in this experiment. If the latest data were not within the upper and lower bounds, then linear interpolation was used to fix the data as follows:

$$\text{fixedData} = \frac{(4 * \text{data}_{n-1} - 2\text{data}_{n-2}) + u + l}{4} \tag{4}$$

where data_{n-1} and data_{n-2} are the previous two data.

Fig. 10(a) shows the prediction results of the initial model. The MAE between real and predicted values is 0.235 °C. Fig. 10(b) compares the MAE of the ACM with a fixed model, which is the same as the initial one of the ACM. The results demonstrate that our ACM always has a better performance than the fixed model, and the gap is 0.014 °C at the second stage and up to 0.168 °C at the last stage. This superiority of our ACM can provide better data cleaning effect. The tested points were divided into four categories: normal, falsely filtered, unfiltered noise, and filtered noise points. Falsely filtered points (ffp) are points that were filtered out but were not noisy points. Unfiltered noise points (unp) were not filtered out but should have been. Filtered noise points (fnp) were noisy and filtered out correctly. Normal points (np) were not noisy and (correctly) not filtered. To clearly compare the data cleaning results between the ACM and the fixed model, we define the following three ratios.

Definition 2 (Falsely filtered ratio): $\text{np}/(\text{ffp} + \text{np})$, which reflects the degree of correct in normal points in the method.

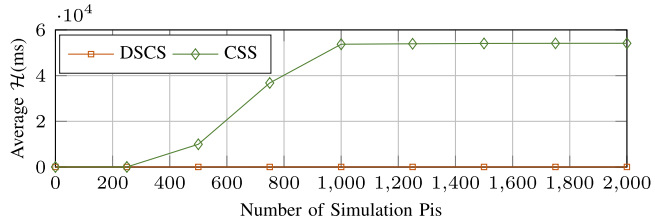

Fig. 12. Average $\mathcal{H}$ versus the number of Pis in the DSCS and CCS with the LSTM-based cleaning model.

Definition 3 (Noise filtered ratio): $\text{fnp}/(\text{fnp} + \text{unp})$, which reflects the ratio of noisy points filtered successfully.

Definition 4 (Filtered ratio): $\frac{\text{fnp}/(\text{fnp}+\text{unp})}{\text{ffp}/(\text{ffp}+\text{np})}$, which reflects the ability to comprehensively filter the data.

The cleaning results are shown in Fig. 11. Fig. 11(a)–(c) is falsely filtered, noise filtered, and filtered ratios, respectively. Note that for all three ratios, the bigger the better. It is clear that the proposed ACM is better than the fixed model because of the dynamic optimization characteristic of our data cleaning system. We also tested $\mathcal{H}$ in this scenario (see Fig. 12). Here, the number of Pis increased with an interval of 250, and $\mathcal{H}$ is the average time of 90-time processes. $\mathcal{H}$ of the DSCS is almost 50 ms, whereas the worst $\mathcal{H}$ of the CCS is 54 s.

At last, to compare DSCS with systems that only use edge devices, we did experiments on processing time considering both training and inference stages of the ACM. Raspberry Pi 4B and Pi 3B were chosen as the edge devices. In the comparison methods, without cloud collaboration, training and inference stages both

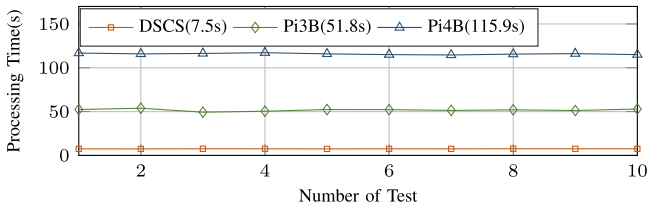

Fig. 13. Comparisons between DSCS and systems that only use edge devices.

run in the edge devices. Fig. 13 shows the results that the DSCS can finish the whole process about 7.5 s, whereas because of the limited resources of edge devices, the average processing time on Raspberry Pi 4B and 3B are 51.8 and 115.9 s, respectively.

C. Discussion of the Two Experiments

From the analyses of aforementioned two experiments, we can observe that the DSCS can adapt well to different scenarios. Meanwhile, it shows superiority over the CCS with less processing time and better cleaning effect. Overall, the main purpose of the DSCS is to provide a unified data cleaning mode for heterogeneous data processing. It particularly applies to scenarios taking the data acquisition and structure into consideration together. Without loss of generality, the DSCS can be installed in any DAS as a module.

IV. Conclusion

In this article, we proposed a data stream cleaning system for the IoT, named the DSCS; it contains a dynamic protocol interpreter, a structure parser, and a cleaning model activator. The dynamic protocol interpreter provides the ability to access a massive heterogeneous data source; the structure parser can arrange the raw data into a specific structure; and the cleaning model activator activates cleaning models. Moreover, the cleaning model can be adaptively supported by a cloud server. To validate the performance of the DSCS, we conducted two experiments: monitoring an injection modeling machine and monitoring base stations. The comparison of $\mathcal{H}$ between the DSCS and a CCS showed that the DSCS can have a stable processing time when the number of accessed Pis increases. Furthermore, because of the adaptive characteristic of the cleaning models in the DSCS, the cleaning effect also increases from the perspective of falsely filtered, noise filtered, and filtered ratios.

In future work, we intend to integrate some real-time databases into our systems, such as BTrDB [27] and EdgeDB [28], and for knowledge development across scenarios in data cleaning, some techniques, such as ontology [29], will also be incorporated.

References

[1] R. Joshi, H. Jadav, A. Mali, and S. Kulkarni, "IoT application for real-time monitor of PLC data using EPICS," in *Proc. IEEE Int. Conf. Internet Things Appl.*, 2016, pp. 68–72.

[2] S. Bao, H. Yan, Q. Chi, Z. Pang, and Y. Sun, "FPGA-based reconfigurable data acquisition system for industrial sensors," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1503–1512, Aug. 2017.

[3] H. Wu *et al.*, "Dynamic edge access system in IoT environment," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2509–2520, Apr. 2020.

[4] OMRON, "Cp1L CPU unit operation manual," pp. 417–506, Oct. 2014, Accessed: May 24, 2021. [Online]. Available: https://assets.omron.eu/downloads/manual/en/v2w462_cp1l_cpu_unit_operation_manual_en.pdf

[5] C.-W. Lin and T.-P. Hong, "A new mining approach for uncertain databases using CUFP trees," *Expert Syst. Appl.*, vol. 39, no. 4, pp. 4084–4093, 2012.

[6] C. K. Leung, R. K. MacKinnon, and S. K. Tanbeer, "Tightening upper bounds to the expected support for uncertain frequent pattern mining," *Procedia Comput. Sci.*, vol. 35, pp. 328–337, 2014.

[7] G. Lee and U. Yun, "A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives," *Future Gener. Comput. Syst.*, vol. 68, pp. 89–110, 2017.

[8] A. Haque, L. Khan, and M. Baron, "SAND: Semi-supervised adaptive novel class detection and classification over data stream," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1652–1658.

[9] L. P. Verma and M. Kumar, "An adaptive data chunk scheduling for concurrent multipath transfer," *Comput. Standards Interfaces*, vol. 52, pp. 97–104, 2017.

[10] M. N. Garofalakis *et al.*, "Probabilistic data management for pervasive computing: The data furnace project," *IEEE Data Eng. Bull.*, vol. 29, no. 1, pp. 57–63, Jan. 2006.

[11] J. I. Maletic and A. Marcus, "Data cleansing: A prelude to knowledge discovery," in *Data Mining and Knowledge Discovery Handbook*. Berlin, Germany: Springer, 2009, pp. 19–32.

[12] Y. Ma, Y. He, A. Way, and J. van Genabith, "Consistent translation using discriminative learning: A translation memory-inspired approach," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2011, pp. 1239–1248.

[13] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek, "AMIE: Association rule mining under incomplete evidence in ontological knowledge bases," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 413–422.

[14] L. Kong, M. Xia, X.-Y. Liu, M.-Y. Wu, and X. Liu, "Data loss and reconstruction in sensor networks," in *Proc. IEEE INFOCOM*, 2013, pp. 1654–1662.

[15] S. Xiang, L. Yuan, W. Fan, Y. Wang, P. M. Thompson, and J. Ye, "Multi-source learning with block-wise missing data for Alzheimer's disease prediction," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2013, pp. 185–193.

[16] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas, "Guided data repair," *Proc. VLDB Endowment*, vol. 4, no. 5, pp. 279–289, 2011.

[17] S. Hao, N. Tang, G. Li, J. He, N. Ta, and J. Feng, "A novel cost-based model for data repairing," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 4, pp. 727–742, Apr. 2017.

[18] C. Ye, H. Wang, K. Zheng, J. Gao, and J. Li, "Multi-source data repairing powered by integrity constraints and source reliability," *Inf. Sci.*, vol. 507, pp. 386–403, 2020.

[19] B. Tang *et al.*, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Trans. Ind. Informat.*, vol. 13, no. 5, pp. 2140–2150, Oct. 2017.

[20] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 2004–2017, Sep. 2018.

[21] H. Wu, J. Hu, J. Sun, and D. Sun, "Edge computing in an IoT base station system: Reprogramming and real-time tasks," *Complexity*, vol. 2019, 2019, Art. no. 4027638.

[22] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar, and J. Stefa, "Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: Review, challenges, and a case study," *IEEE Netw.*, vol. 30, no. 2, pp. 54–61, Mar./Apr. 2016.

[23] X. Chen, Q. Shi, L. Yang, and J. Xu, "ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications," *IEEE Netw.*, vol. 32, no. 1, pp. 61–65, Jan./Feb. 2018.

[24] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4712–4721, Oct. 2018.

[25] R. M. Hierons and U. C. Türker, "Parallel algorithms for testing finite state machines: Generating UIO sequences," *IEEE Trans. Softw. Eng.*, vol. 42, no. 11, pp. 1077–1091, Nov. 2016.

[26] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," in *Proc. 2016 Conf. Empirical Methods Nat. Lang. Process.*, Austin, TX, USA, 2016, pp. 551–561, doi: 10.18653/v1/D16-1053.
[27] M. P. Andersen and D. E. Culler, "BTrDB: Optimizing storage system design for timeseries processing," in *Proc. 14th USENIX Conf. File Storage Technol.*, 2016, pp. 39–52.
[28] Y. Yang, Q. Cao, and H. Jiang, "EdgeDB: An efficient time-series database for edge computing," *IEEE Access*, vol. 7, pp. 142 295–142307, 2019.
[29] F. Zablith *et al.*, "Ontology evolution: A process-centric survey," *Knowl. Eng. Rev.*, vol. 30, no. 1, pp. 45–75, 2015.

Danfeng Sun (Member, IEEE) received the M.S. degree in computer architecture from Hangzhou Dianzi University, Hangzhou, China, in 2014.

He is currently a Research Assistant with the Institute of Industrial Internet, Hangzhou Dianzi University. His research interests include embedded systems, machine learning, and the Industrial Internet of Things.

Shan Xue received the Ph.D. degree in computer science from the Center for Artificial Intelligence, University of Technology Sydney, Ultimo, NSW, Australia, and the Ph.D. degree in information management from the School of Management, Shanghai University, Shanghai, China.

She is currently a Research Fellow with the Department of Computing, Faculty of Science and Engineering, Macquarie University, Sydney, NSW, as well as a Researcher of CSIRO's Data61, Sydney. Her current research interests include artificial intelligence, machine learning, and knowledge mining.

Huifeng Wu (Member, IEEE) received the Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2006.

He is currently a Professor with the Institute of Intelligent and Software Technology, Hangzhou Dianzi University, Hangzhou. His research interests include software development methods and tools, software architecture, embedded systems, intelligent control and automation, and the Industrial Internet of Things.

Jia Wu (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Technology Sydney, Ultimo, NSW, Australia.

He is currently an ARC DECRA Fellow with the Department of Computing, Macquarie University, Sydney, NSW. Since 2009, he has authored or coauthored 100+ refereed journal and conference papers, including IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON MULTIMEDIA, *ACM Transactions on Knowledge Discovery from Data* (TKDD), *Neural Information Processing System*, International World Wide Web Conference, and ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. His current research interests include data mining and machine learning.

Dr. Wu was the recipient of SDM'18 Best Paper Award in Data Science Track, IJCNN'17 Best Student Paper Award, and ICDM'14 Best Paper Candidate Award. He is the Associate Editor for the ACM TKDD and *Neural Networks*.