The research of data conflict in digital camp management and control system based on IOT

Xingchen Li[1*], Shenglin Li[2], Heng Zhang[2], Hui Cai[1]

[1]*Graduate school, Logistical Engineering University of P.L.A, Chongqing, China*

[2]*Department of Information Engineering, Logistical Engineering University of P.L.A, Chongqing, China*

Received 1 May 2014, www.cmnt.lv

Abstract

A core technology in IOT (Internet of Thing) is data processing and there may be a mismatch between data collection and data interface. Aiming at data conflicts in the DCMCS (Digital Camp Management Control System), this paper defines data conflicts with formal description and divides the data conflicts of the DCMCS into six types. For each type of the data conflicts, a resolution strategy is designed to solve it. Then the paper designs the DCRS (Data Conflict Resolution System) to detect and resolve the data conflicts automatically and quickly. The solutions provide technical support for data interaction, which have been successfully verified in the DCMCS.

Keywords: formal description, data conflict, resolution strategy, DCRS

1 Introduction

As an important goal of modern barracks to transform information, the DCMCS (Digital Camp Management Control System) refers to utilize sensing technology, network communications and automatic control technology to build IOT with barracks facilities and equipment, getting the operational status of devices in real time and controlling them accurately. In the process of collecting data, there may be a mismatch between data acquisition and the upper data interface due to sensor noise, external interference, device aging or deviation of application itself and resultant data conflicts. Therefore, it is important to research data conflicts in the DCMCS and resolution method.

2 Backgrounds

A core technology in IOT is data processing. Data processing in IOT can be divided into data recognition and data interaction [1]. DCMCS data conflict is the data exchange errors arising from mismatches between data interface and device identification, data types, data structures, data accuracy and acquisition space. That is, when output data of sensing devices cannot meet the restrictions of the data input interface, it will produce data conflict. For example, sensor D1 outputs float-type data and integer-type is required by the interface D2, so it will generate data type conflict.

Current research of data Processing in IOT is about heterogeneous data storage and abnormal data detection. In terms of storage and query processing of sensor sampling data, distributed storage method [2-4] means that sample data are stored directly in each sensor node or in the sink node, and are obtained through remote access query processing. If query processing involves different sensor networks, it needs to be converted by the middleware to achieve interoperability operation between heterogeneous sensor networks [5, 6]. With respect to centralized processing of sensor data, the most direct way is to use cloud data management and related technologies. Currently a majority of cloud data management systems are the "key-value" database such as Bigtable [7], Dynamo [8], Hbase [9] and so on. "Key-value" database can efficiently handle queries based on the primary key, but fails to support networked data storage, spatial and temporal logic query and attribute constraints inquiries. Parallel database technology [10, 11] supports the processing of structured data by organizing multiple relational databases into a massive database cluster, but the method cannot retrieve quickly the required data based on the identity of the sensor. Common methods of anomaly data detection are statistics, feature selection, neural networks, data mining and wavelet singularity detection. The main idea based on statistics [12] is to assume that data sets obey certain distribution or probability model and that serious deviation from the distribution curve object are recorded as outliers by calculating the probability of the object in the distribution model. Outlier based on distance was first proposed by the Knor and Ng [13], who viewed the records as points in data space. Outlier is defined as the distance between the point and most points of data sets, which are larger than the value of a certain point.

Literature reveals that credibility or a statistical model would normally be added in the uncertain data of sensors to detect abnormal data. Data conflicts due to a mismatch between sensor data acquisition and the interface need to be researched further. In view of this, the paper proposes a

***Corresponding author** e-mail: since0701@163.com

data conflict detection method based on formal description, and then puts forward a conflict resolution program to quickly determine and digest data conflict in the environment of mass data.

3 Formal description of data conflict

Formal description is the method of describing the system properties with the mathematical basis formal symbol for the specification, design and validation of computer systems. It can accurately and succinctly describe phenomena, objects or action results and is ideal for modelling tools. Samples, data conflicts, data interfaces and interface matching principles are described and defined in this section. The accurately formal description of data conflicts can provide good support for the digestion of data conflict and provide a basis for the development of the DCRS.

The DCMCS needs to read the device identification and sampling data, including sampling value, and spatial and temporal information, such as sampling time and sampling location, which reflect the characteristics of the target object. Sample sequence can be composed of a number of samples of the device. Due to the heterogeneity of collection data, a uniform representation is needed for samples which sensing devices collected with unique identity, time, location, sample values and units of measurement.

Definition 1 (The Definition of Sample-Value): In the DCMCS, Sample-Value is expressed as $S =< ID, T, L, V, U >$, where *ID* is the unique identification of sampling device; *T* is the acquisition time; *L* represents the sampling location. For a stationary device, sampling location is the place that the sensor is installed and for mobile devices, it means the places where collect action takes place; *V* is the value of sensor data collection; *U* is the measure unit of data value. S_i shows the sample data of i^{th} device.

After data collection, the sensor needs to exchange the data with the data management system through data interface to process data collected from the sensors.

Definition 2: (The Definition of Data Interface) In the DCMCS, the data interface of the data management system is expressed in a ternary suit $A =< I, G, D >$, $I = \{I_1, I_2 \cdots I_n\}, (n \in N)$ indicate the function set of the data interface. $I_i (i \in N, 1 \le i \le n)$ show the i^{th} interface of the data interface; $G = \{G_1, G_2 \cdots G_n\}, (n \in N)$ indicate the functional function set of the data interface, G_i show the set of functional function corresponding with interface I_i; $D = \{D_1, D_2 \cdots D_n\}, (n \in N)$ indicate the set of input and output data of the data interface, and D_i show the set of input and output data corresponding with interface I_i.

Definition 3: (The Definition of Input/Output of Data Interface) In the data interaction, the data interface $I =< I^{in}, I^{out} >$, $I^{in} = \{P_1, P_2 \cdots P_m\}, (m \in N)$ indicate the input of data interface. $I^{out} = \{P_1, P_2 \cdots P_n\}, (n \in N)$ indicate the output of data interface, and $P =< M, F, J, R >$ show the parameters of input/output and restriction conditions of those parameters, where *M* indicate the types of input/output parameters, *F* indicate the structure of input/output parameters, *J* indicate the precision of input/output data, *R* indicate the value range of input/output data. $P_i (i \in n)$ indicates the i^{th} parameter and its restriction conditions of input/output interface *I*.

Definition 4: (The Definition of Data Interface Matching) Interface $A_i =< I_i, G_i, D_i >$, where $I_i =< I_i^{in}, I_i^{out} >$, the input of interface $I_i^{in} = P_i^{in} =< M_i^{in}, F_i^{in}, J_i^{in}, R_i^{in} >$, the output of sensor $I_i^{out} = P_i^{out} =< M_i^{out}, F_i^{out}, J_i^{out}, R_i^{out} >$, if $\exists (P_i^{out} = P_i^{in})(i \in n)$, output data I^{out} matches input data I^{in}, denoted as $I^{out} \Rightarrow I^{in}$.

Definition 5: (The Definition of Data Conflict) Define $H_{A_i} = \begin{cases} 1, A_i \ Normal \ \text{interaction} \\ 0, A_i \ Abnormal \ \text{interaction} \end{cases}, (i \in N)$. In the DCMCS $\exists A_i$, if $P_i^{out} \ne P_i^{in}$, $I^{out} \Rightarrow I^{in}$ is not established, we call it 'data conflict' and $H_{A_i} = 0 (i \in N)$ at this time.

4 Types of data conflict

Data conflicts of the DCMCS based on IOT are mainly caused by the identity of the sensor, the fact that the output data does not match the constraints of the interface and so forth. Data conflicts can be summarized as the following six types: *ID* conflict, data type conflict, data structure conflict, data precision conflict, data overflow conflict and acquisition space conflict, as shown in Figure 1.

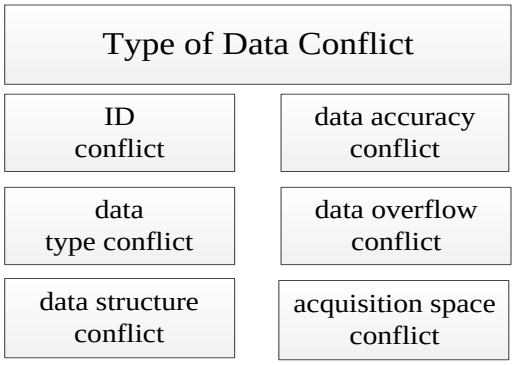

FIGURE 1 The type of data conflict

ID conflict: The IOT of digital camps has a lot of sensor nodes and it needs to encode these sensing devices and give them unique *ID* for identification. During the dynamic process of adding or deleting devices, it may encode different sensors with the same *ID*. The system cannot locate the unique sensing device and then generates data conflict. In a single round of data collection process, $\exists A_i =< I_i, G_i, D_i >$, $\exists A_j =< I_j, G_j, D_j >$, where $D_i =< ID_i, T_i, L_i, V_i, U_i >$, $D_j =< ID_j, T_j, L_j, V_j, U_j >$. If $ID_i =$

ID_j, we call it *ID* conflict and denote as $Conf_{ID}(IDi, IDj)$. *ID* conflict occurs when samples that have the same *ID* try to join sample sequences, causing the collect data not be received normally by the data interface.

Data type conflict: As it cannot transfer impliedly in data interaction, data type conflict will occur if the data types of input and output are inconsistent and lead to errors in data exchange. $\exists A_i =< I_i, G_i, D_i >$, where $I_i =< I_i^{in}, I_i^{out} >$, $I_i^{in} = P_i^{in} =< M_i^{in}, F_i^{in}, J_i^{in}, C_i^{in} >$, $I_i^{out} = P_i^{out} =< M_i^{out}, F_i^{out}, J_i^{out}, C_i^{out} >$, if $M_i^{in} \neq M_i^{out}$, then $P_i^{in} \neq P_i^{out}$, that means $I^{out} \Rightarrow I^{in}$ is not established and makes $H_{A_i} = 0$. We call the data conflict as data type conflict and denoted as $Conf_{type}(DType1, DType2)$.

Data structure conflict: Data structure is a kind of type set and it is a plurality of attributes including the number of parameters and data types. *DStruct1* and *DStruct2* could be the structure types defined by system or itself, which represent the output data structure and the input data structure. If there is an inconsistent type in *DStruct1* and *DStruct2*, data structure conflict appears. The interface cannot recognize or accept the data collected. $\exists A_i =< I_i, G_i, D_i >$, where $I_i =< I_i^{in}, I_i^{out} >$, $I_i^{in} = P_i^{in} =< M_i^{in}, F_i^{in}, J_i^{in}, C_i^{in} >$, $I_i^{out} = P_i^{out} =< M_i^{out}, F_i^{out}, J_i^{out}, C_i^{out} >$, if $F_i^{in} \neq F_i^{out}$, then $P_i^{in} \neq P_i^{out}$, that means $I^{out} \Rightarrow I^{in}$ is not established and makes $H_{A_i} = 0$. We call it data structure conflict, denoted as $Conf_{struct}(DStruct1, DStruct2)$.

Data accuracy conflict: Data accuracy will directly affect the result of data calculation. If the precision of input data and output data is inconsistent, for example the output data of sensor is float while the interface needs integer data, a data conflict generates and the interface cannot get correct data. $\exists A_i =< I_i, G_i, D_i >$, where $I_i =< I_i^{in}, I_i^{out} >$, $I_i^{in} = P_i^{in} =< M_i^{in}, F_i^{in}, J_i^{in}, C_i^{in} >$, $I_i^{out} = P_i^{out} =< M_i^{out}, F_i^{out}, J_i^{out}, C_i^{out} >$, if $J_i^{in} \neq J_i^{out}$, then $P_i^{in} \neq P_i^{out}$, that means $I^{out} \Rightarrow I^{in}$ is not established and makes $H_{A_i} = 0$. We call the data conflict as accuracy conflict, denoted as $Conf_{Accur}(DAccur1, DAccur2)$.

Data overflow conflict: In the data exchange process, sensor collecting data exceeding the defined range of data interface and generating value range conflict is a kind of normal phenomenon. For example, the original collecting data of pressure sensor is 42560 and the data interface is defined as 0-42559 corresponding to a pressure of 0-1Mpa, then data conflict occur. $\exists A_i =< I_i, G_i, D_i >$, where $I_i =< I_i^{in}, I_i^{out} >$, $I_i^{in} = P_i^{in} =< M_i^{in}, F_i^{in}, J_i^{in}, C_i^{in} >$, $I_i^{out} = P_i^{out} =< M_i^{out}, F_i^{out}, J_i^{out}, C_i^{out} >$, $D_i =< ID_i, T_i, L_i, V_i, U_i >$, if $V_i \notin R_i$, then $P_i^{in} \neq P_i^{out}$, that means $I^{out} \Rightarrow I^{in}$ is not established and makes $H_{A_i} = 0$. We call the data conflict as data overflow conflict, denoted as $Conf_{range}(DRange1, DRange2)$.

Acquisition space conflict: Due to the spatial and temporal characteristics of the data collected, sensors can only collect data in one place at the same time. In a single round of the data collection process, $\exists Ai =< Ii, Gi, Di >$, $\exists A_j =< I_j, G_j, D_j >$, where $D_i =< ID_i, T_i, L_i, V_i, U_i >$, $D_j =< ID_j, T_j, L_j, V_j, U_j >$. If $L_i = L_j$, we call the data conflict as acquisition space conflict, denoted as $Conf_{Loc}(Loc1, Loc2)$.

5 Strategies to resolve data conflict

On basis of formal description, we divide data conflicts into six categories: *ID* conflict, data type conflicts, data structure conflict, data accuracy conflict, data overflow conflict and acquisition space conflict. Hence, there are also six strategies to resolve data conflicts, as shown in this section.

Digestion strategies for *ID* conflict: When an *ID* conflict occurs, rename the sensor according to the naming rule that the acquisition time goes behind the other sensor, in order to make the identity different. The implementation procedures are, first, to determine the identity of the two sensors. If they are inconsistent, data can be directly released; otherwise compare the acquisition time of samples T_1 and T_2. If $T_1 > T_2$, then rename ID_1 according to the naming rule; if $T_1 < T_2$, then rename ID_2. The realization function is: private SensorData ReID(SensorData data1) Figure 2:

```
1 public class SensorData
2     {
3     private string _id;
4     public string Id
5     {
6     get { return _id; }
7     set { _id = value; }
8     }
9     private DateTime _measuringTime
10    public DateTime MeasuringTime
11    {
12    get { return _measuringTime; }
13    set { _measuringTime = value; }
14    }
15    }
16    private SensorData ReID(SensorData data1)
17    {
18    int count = 0;
19    SensorData ReSensorData = new SensorData();
20    ReSensorData = data1;
21    foreach (SensorData data in sensorDatas)
22    {
23    if (data.Id.Equals(data1.Id))
24    count++;
25    if (count > 1 && (data.MeasuringTime >
      data1.MeasuringTime))
26    ReSensorData = data;
27    }
28    return ReSensorData;
29}
```

FIGURE 2 The source code of digestion strategies for *ID* conflict

Digestion Strategies for data type conflict: Converting the data type of the sensor output to make it convert with

the input of interface. The implementation procedures are, first, to judge the consistency of output data type $DType_1$ and interface data type $DType_2$. If they are consistent, data can be directly released; otherwise, it needs to transfer the type of sensor output data $Dtype_1$ according to the type of interface data. So that it can meet the constraints of the interface, like Integer transfer to String or String transfer to Integer, since the data interfaces are public and can be realized with middleware, database management systems and other forms. The realization function is: private SensorData ReDataType(SensorData data1) Figure 3:

```
1 public class SensorData
2   {
3   private string _dataType;
4   public string DataType
5   {
6   get { return _dataType; }
7   set { _dataType = value; }
8   }
9   private string _dataValue;
10  public string DataValue
11  {
12  get { return _dataValue; }
13  set { _dataValue = value; }
15  }
16  }
17  private SensorData ReDataType
          (SensorData data1)
18  {
19  SensorData ReSensorData = new SensorData();
20  Boolean success = true;
21  try
22  {
23   switch (data1.DataType)
24  {
25  case "int":
26   Convert.ToInt32(data1.DataValue); break;
27   case "float":
28  double.Parse(data1.DataValue); break;
29   case "date":
30  Convert.ToDateTime(data1.DataValue); break;
31   default: break;
32  }
33  }
34  catch
35  {
36  success = false;
37  }
38  if (success)
39  return ReSensorData;
40  else
41  {
42  return chooseOtherDataValue(data1.Id);
43  }
44  }
```

FIGURE 3 The source code of digestion strategies for data type conflict

Digestion Strategies for data structure conflict: Converting structure of the sensor output data in order to make it convert with the structure of interface input data. The implementation procedures are, first, to compare the structure of the sensor output data $Dstructure_1$ with the structure of interface input data $Dstructure_2$; analyse the consistency of the structures of two samples, including the number of acquisition parameters, data type and other attributes. Then transfer $Dstructure_1$ consistent with $Dstructure_2$ which is not mismatched to make it meet the constraints of the interface. The realization function is: private SensorData ReDataStruct(SensorData data1) Figure 4:

```
1 public class SensorData
2   {
3   private string _dataStruct;
4   public string DataStruct
5   {
6   get { return _dataStruct; }
7   set { _dataStruct = value; }
8   }
9   private string _address;
10  public string Address
11  {
12  get { return _address; }
13  set { _address = value; }
14  }
15  }
16  private SnsorData ReDataStruct(SensorData data1)
17  {
18  if (!data1.DataStruct.Split('-')[1].Equals("1"))
19  {
20  IList<int> count=new  List<int>();
21  foreach (SensorData data in sensorDatas)
22  {
23  string[] numbers = data1.DataStruct.Split('-');
24  if (data.Address.Equals(data1.Address))
25  {
26  count.Add(int.Parse(numbers[0]));
27  }
28  }
29  count.Add(int.Parse(data1.DataStruct.
    Split('-')[0]));
30  bool repeat=true;
31  if (count.Count ==  int.Parse(data1.DataStruct.Split('-')[1]))
32  {
33  for (int i = 0; i < count.Count - 1; i++)
34  {
35  for (int j = 0; j < count.Count - 1; j++)
36  {
37  if (count[i] == count[j] && i != j)
38   repeat = true;
39  }
40  }
41  if(repeat)
42  {
43  return chooseOtherDataValue(data1.Id);
44  }
45   else return data1;
46  }
47   else
48  {
49  return chooseOtherDataValue(data1.Id);
50  }
51  }
52  else
53  return data1;
54   }
```

FIGURE 4 The source code of digestion strategies for data structure conflict

Digestion Strategies for data accuracy conflict: The digestion strategies for data accuracy conflict are to unite the accuracy of sensor output data and the accuracy of interface input data with the same constraints. The implementation procedures are to calibrate $DAccuracy_1$ according to the accuracy of interface $DAccuracy_2$. For values larger than accuracy, use the rounded method and for those smaller than accuracy, use the padding method at the end of zero. The realization function is: private SensorData ReDataAccuracy(SensorData data1) Figure 5:

```
1 public class SensorData
2         {
3         private string _dataAccuracy;
4         public string DataAccuracy
5         {
6         get { return _dataAccuracy; }
7         set { _dataAccuracy = value; }
8         }
9         private string _dataValue;
10        public string DataValue
11        {
12        get { return _dataValue; }
13        set { _dataValue = value; }
14        }
15        }
16      private SensorData ReDataAccuracy(SensorData data1)
17        {
18        if (data1.DataAccuracy.Equals(""))
19        return data1;
20        else
21         {
22        if (data1.DataAccuracy.Contains('.'))
23        {
24        int accuracy =
25        Convert.ToInt32(data1.
          DataAccuracy.Split('.')[0]);
26        if (data1.DataValue.Length <= accuracy)
27        {
28        int accuracy1 = Convert.ToInt32(data1.
          DataAccuracy.Split('.')[1]);
29        if (data1.DataValue.Length <= accuracy1)
30        {
31        return data1;
32        }
33        else
34        {
35        return chooseOtherDataValue(data1.Id);
36        }
37        }
38        else
39        {
40        return chooseOtherDataValue(data1.Id);
41        }
42        }
43        else
44        {
45        int accuracy = 31Convert.ToInt32(data1.
          DataAccuracy);
46        if (data1.DataValue.Length <= accuracy)
47        {
48        return data1;
49        }
50        else
51        {
52         return chooseOtherDataValue(data1.Id);
53        }
54        }
55        }
56 }
```

FIGURE 5 The source code of digestion strategies for data accuracy conflict

Digestion Strategies for data overflow conflict: The digestion strategies for data overflow conflict are to limit the value range of input/output $DRange_1$ and $DRange_2$ to exclude the overflow data. The implementation procedures are that the value range of input/output is $DRange1 \cap DRange2$. If $DRange1 \cap DRange2 = \emptyset$, the sample can't interact with the interface. System alarms to remind invalid data and the sensor output value range needs to be defined so that it does not exceed the value range. The realization function is: private SensorData ReDataRange(SensorData data1) Figure 6:

```
1 public class SensorData
2         {
3         private string _dataValue;
4         public string DataValue
5         {
6         get { return _dataValue; }
7         set { _dataValue = value; }
8         }
9         private string _minData;
10        public string MinData
11        {
12        get { return _minData; }
13        set { _minData = value; }
14        }
15        private string _maxData;
16        public string MaxData
17        {
18        get { return _maxData; }
19        set { _maxData = value; }
20        }
21        }
22        private SensorData ReDataRange(SensorData data1)
23        {
24        if (data1.DataType.Equals("int") ||
          data1.DataType.Equals("float"))
25        {
26      if (double.Parse(data1.DataValue) >=
        double.Parse(data1.MinData) &&
        double.Parse(data1.DataValue) <=
        double.Parse(data1.MaxData))
27        {
28        return data1;
29        }
30        else
31        {
32        alert();
33        return chooseOtherDataValue(data1.Id);
34        }
35        }
36        else return data1;
37        }
```

FIGURE 6 The source code of digestion strategies for data overflow conflict

Digestion Strategies for acquisition space conflict: The digestion strategies for acquisition space conflict are to correct the data acquisition location. The implementation procedures are to determine the type of the sensor when an acquisition space conflict occurs. If it is a mobile device, then release the data; For a fixed device, modify the error collection address according to the device installation table that records the installation site. Its function is: private SensorDataReDataLocation (SensorData data1) and private SensorData chooseOtherDataValue(string id) Figure 7:

```
1 public class SensorData
2         {
3         private int _deviceType;
4         public int DeviceType
5         {
6         get { return _deviceType; }
7         set { _deviceType = value; }
8         }
9         private string _location;
10        public string Location
11        {
```

```
12        get { return _location; }
13        set { _location = value; }
14        }
15        }
16        private SensorData ReDataLocation
         (SensorData data1)
17        {
18        if (data1.DeviceType.Equals(0))
19        {
20        if(data1.Location.Equal
         (getLocation(data1.Id)))
21        {
22        return data1;
23        }
24        else
25        {
26        return chooseOtherDataValue(data1.Id);
27        }
28        }
29        else return data1;
30        }
31        private SensorData choose
         OtherDataValue(string id)
32        {
33        return new SensorData();
34        }
```

FIGURE 7 The source code of digestion strategies for acquisition space conflict

6 Realization of data conflict resolution system

On the basis of the formal description of data conflict and strategies to solve data conflict, the paper designs the Data Conflict Resolution System in this section to detect and solve the data conflict automatically and quickly. DCRS aims at adopting relevant strategies for each sort of data conflict to realize the goal of resolving conflicts while at the same time increasing system stability and reliability. If the input and output of data interface does not match, the DCRS calls the conflict resolution to deal with invalid data. Its framework flow is showed in Figure 8.

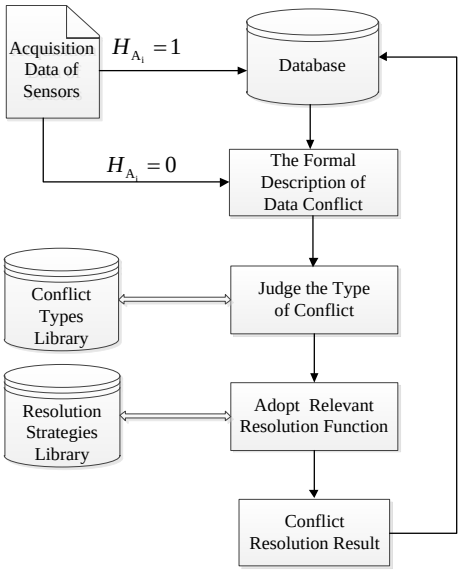

FIGURE 8 The framework flow of DCRS

In the framework flow of the DCRS, conflict type library has six data conflicts and resolution strategies library has resolution function corresponding to the conflicts. Firstly, the DCRS detects data conflicts in the input and output of the interface. If the samples match the constraints of the interface, the DCRS releases data into database; otherwise，the DCRS identifies the type of the data conflict according to the formal description of the conflict type library, and then calls the appropriate function to resolve conflicts. After that, the DCRS releases the data into database.

The DCRS adopts Microsoft. NET platform for developing and data conflict digestion algorithm uses the technology of C# language. Visual Studio. NET is Microsoft's second-generation development tool for building and deploying powerful and secure software. The database system of the DCRS uses Oracle 10g, the relational database management system with features of large data capacity, persistent storage time, convenient data sharing, and high reliability. The database system is used to store the conflict type and dynamic data generated during the system operation. The main interface of the DCRS is shown in Figure 9.

FIGURE 9 The main interface of DCRS

There are a variety of criteria to evaluate algorithm performance, and what kind of standard is adopted is mainly depended on the system requirements. In the DCMS real-time is the most important factor, and therefore the processing time overhead and memory space overhead can be used to evaluate the performance of the data conflict resolution algorithm. Processing time overhead is the time that digestion algorithm takes to process the data conflict. Transaction will be worthless to the system if it is completed after the zero-value point and is a waste of system resources. Memory space overhead is the memory occupied by the algorithm when it digests the data conflicts and is used to store the information generated in the process. Algorithm that is rational designed should effectively utilize system resources to process the data conflict without taking up too many resources. The PC that the DCRS runs on with CPU clock at 2.5GHz and 4GB memory is used to test the algorithm performance. The performance test uses one hundred thousand data collected by the water meters and electric meters in the DCMCS. There are 181 abnormal data when

the test finishes and it costs 2505ms to resolute data conflicts. Among the data conflicts, there are 7 ID conflicts that need 31.5ms to process, 4.5ms for each conflict. There are 35 data type conflicts that need 318.5ms to process, 9.1ms each. There are 28 data structure conflicts that need 336ms to process, 12ms each. There are 53 data accuracy conflicts that need 726.1ms to process, 13.7ms each. There are 46 data overflow conflicts that need 230ms to process, 5ms each. There are 12 acquisition space conflict that need 75.6ms to process, 5.3ms each. The processing time overhead for the six types of conflict digestion algorithm is showed in Figure 10.

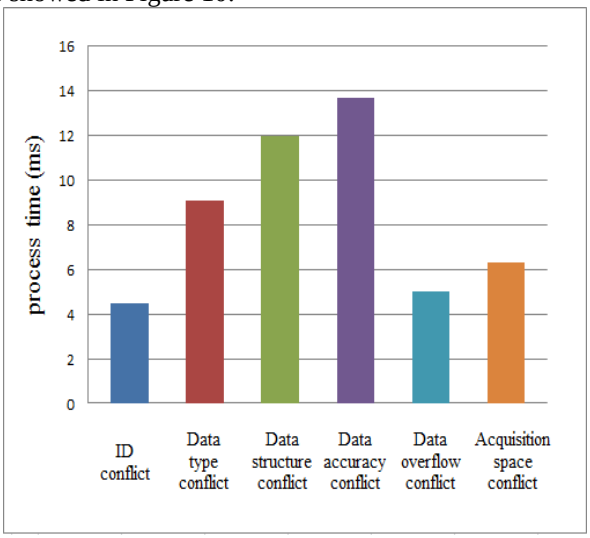

FIGURE 10 The processing time overhead for the six types of conflict digestion algorithm

The memory space overhead for the six types of conflict digestion algorithm is shown in Figure 11. It can be seen from the test that the digestion algorithm can meet the real-time data processing system requirements.

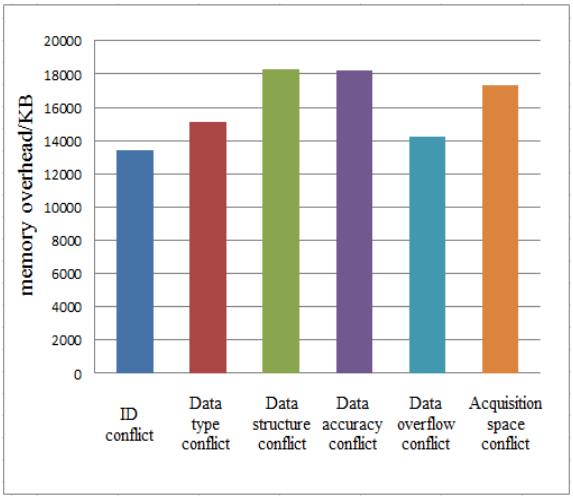

FIGURE 11 The memory space overhead for the six types of conflict digestion algorithm

7 Conclusions

In the DCMCS, there are data conflicts caused by a lack of uniform standards for interaction between the device of IOT and the upper data interface. To address this problem, the paper analyses formal description of data conflict, proposes strategies to resolve conflict, designs resolution flow and exploits the DCRS. The solutions have been successfully verified in the DCMCS, realizing intellectualized resolution of data conflict and providing technical support and insightful guarantee for data interaction.

Acknowledgment

Fund project: the army logistics key scientific research program funded projects of PLA(BS211R099).

References

[1] Hu Y L, Sun Y F, Yin B C 2012 Information sensing and interaction technology in Internet of Thing *China Journal of Computers* **35**(6) 1147-63 *(in Chinese)*
[2] Tsiftes N, Dunkels A 2011 A database in every sensor *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems Seattle USA* 316-29
[3] Chen X, He X, Guo H, Wang Y 2011 Design and Evaluation of an Online Anomaly Detector for Distributed Storage Systems *Journal of Software* **6**(12) 2379-90 *(in Chinese)*
[4] Madden S, Franklin M J, Hellerstein J M, Hong W, Tiny 2005 DB: An acquisitional query processing system for sensor networks *ACM Transactions on Database Systems* **30**(1) 122-73
[5] Gurgen L, Roncancio C, Labbé C 2008 SStreaMWare: A service oriented middleware for heterogeneous sensor data management *Proceedings of the 5th International Conference on Pervasive Services (ICPS'08) Sorrento Italy* 121-30
[6] Kim M, Lee J W, Ryou J C 2008 COSMOS: A middleware for integrated data processing over heterogeneous sensor networks. *ETRI Journal*, **30**(5) 696-706
[7] Chang F, Dean J, Ghemawat S 2006 Bigtable: A distributed storage system for structured data *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06) Seattle USA* 205-8
[8] DeCandia G, Hastorun D, Jampani M 2007 Dynamo: Amazon's highly available key-value store *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'2007) Washington USA* 205-20
[9] Carstoiu D, Lepadatu E, Gaspar M 2010 Hbase: Non SQL database, performances evaluation *International Journal of Advancements in Computing Technology* **12**(5) 42-52
[10] Zhou Y, Zhu Q, Zhang Y 2011 Spatial Data Dynamic Balancing Distribution Method Based on the Minimum Spatial Proximity for Parallel Spatial Database *Journal of Software* **6**(7) 1337-44
[11] Poess M, Nambiar R O 2005 Large scale data warehouses on grid: Oracle database 10g and HP proLiant systems *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'2005) Trondheim Norway* 1055-66
[12] Zhang W S, Cao G H. 2004 *IEEE Transactions on Wireless Communication* **3**(5) 1689-701
[13] Chen H F, Mineno H, Mizuno T 20080 Adaptive data aggregation scheme in clustered wireless sensor networks *Computer Communications* **31**(15) 3579-8

Authors

Xingchen Li, born in 1986, Sichuan, China

Current position, grades: Ph.D student of the Department of Information Engineering, Logistical Engineering University of P.L.A.
University studies: B.S. and M.S. degrees from Logistical Engineering University of P.L.A in 2008 and 2011 respectively.
Scientific interest: data processing and decision support system.

Shenglin Li, born in 1964, Sichuan, China

Current position, grades: professor of the Department of Information Engineering, Logistical Engineering University of P.L.A.
University studies: Ph.D. degree in Logistical Engineering University of P.L.A China, in 2008.
Scientific interest: information management engineering and cloud computing.

Heng Zhang, born in 1981, Chongqing, China

Current position, grades: lecturer of the Department of Information Engineering, Logistical Engineering University of P.L.
University studies: B.S. and M.S. degrees from Logistical Engineering University of P.L.A in 2004 and 2008 respectively.
Scientific interest: knowledge engineering, modelling and control of complex systems.

Hui Cai, born in 1982, Chongqing, China

Current position, grades: Ph.D. student of the Department of Information Engineering, Logistical Engineering University of P.L.
University studies: M.S. degrees from Logistical Engineering University of P.L.A in 2010.
Scientific interest: knowledge engineering, modelling and control of complex systems.