Research Article

A Dynamic Processing System for Sensor Data in IoT

Minbo Li,1,2 Yanling Liu,1 and Yuanfeng Cai1

1Software School, Fudan University, Shanghai 201203, China
2Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China

Correspondence should be addressed to Minbo Li; limb@fudan.edu.cn

Received 5 December 2014; Revised 23 April 2015; Accepted 1 June 2015

Academic Editor: Xiuzhen Cheng

Copyright © 2015 Minbo Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the development of the Internet of Things (IoT for short), innumerable Wireless Sensor Networks (WSNs) are deployed to capture the information of environmental status in the surrounding physical environment. The data from WSNs, called sensor data, are generated in high frequency. Similar to data of other open-loop applications, for example, network monitoring data, sensor data are heterogeneous, redundant, real-time, massive, and streaming. Hence, sensor data cannot be treated as the IoT business data, which brings complexity and difficulty to information sharing in the open-loop environment. This paper proposes a dynamic sensor data processing (SDP) system to capture and process sensor data continuously on the basis of data streaming technology. Particle Swarm Optimization (PSO) algorithm is employed to train threshold dynamically for data compression avoiding redundancy. With the help of rules setting, the proposed SDP is able to detect exception situations. Meanwhile, the storage models in SQL and NOSQL databases are analyzed and compared trying to seek an appropriate type of database for sensor data storage. The experimental results show that our SDP can compress sensor data through dynamically balancing the accuracy and compression rate and the model on NOSQL database has better performance than the model on SQL database.

1. Introduction

The Internet of Things (IoT) [1–3] is a concept which aims to integrate the virtual world of information technology with the real world of things seamlessly. The perception layer of the IoT consists of a large number of Wireless Sensor Networks (WSNs) [4]. Sensors in WSNs are adopted to sense the environment bridging machines and physical world, especially in manufacturing and logistics applications. Sensors frequently collect heat, power, light, sound, speed, and so on as signals to provide raw information for information systems. These signals are sensor data which reveal the environmental status. In traditional sensor applications, such as greenhouse monitoring, sensor data were processed and stored in a single system. Accordingly, there was little need for exchange of data across organizational boundaries. Hence, the traditional sensor applications often become close-loop applications. The IoT applications, which always involve various stakeholders and require information exchange and sharing, are open-loop applications. Similar to other applications in the open-loop environment, the sensor data in the IoT applications have the following five characteristics.

Sensor Data Are Heterogeneous. The types and functions of sensors are diverse. Temperature and humidity sensors and illumination sensors are common in agricultural and logistic applications, while blood sensors monitor blood pressure, heart rate, and other body indices. Different types of devices monitor various indices. Besides, the protocols the devices use are diverse. The data from various sensors are in heterogeneous formats.

Sensor Data Are Redundant. The state of the environment or the human being whom a sensor is monitoring is stable during a short time. For example, the temperature of a room may remain unchanged or float little in 5 minutes in common situations. Since sensors collect information in seconds, database will meet heavy load, grow quickly, and have a bad performance if storing all of those data.

Sensor Data Are Real-Time Data. Sensor data are collected over time. So sensor data are real-time data in fact. Processing the sensor data in time can bring benefits into the IoT. For example, once an abnormal situation occurs, a timely
exception detection can help operators make decisions or adjust measures fast to reduce the loss.

Sensor Data Are Massive and Streaming. As the production costs of sensors come down, more and more sensors are applied to detect complete environmental information. Meanwhile, sensors are set to collect data in seconds. Hence, sensor data are generated and streamed to the Internet. And the amount of sensor data is large.

In order to maximize the use of sensor data, the processing systems are required to recognize these characteristics. We propose an IoT sensor data processing (SDP) system based on data streaming technology, trying to do dynamic data compression reducing redundancy with the help of Particle Swarm Optimization (PSO) algorithm and to detect exception situations in real-time. Meanwhile, we seek an appropriate type of database to store massive sensor data. The experimental results show that compression method can achieve better performance than the model on SQL database.

The remainder of the paper is organized as follows. Section 2 describes the related work. Section 3 introduces the preliminary notions. Section 4 presents the framework of the proposed processing system SDP. In Section 5, we conducted experiments on real sensor data and present the results. Section 6 concludes this paper.

2. Related Work

Existing researches on sensor data processing can be summarized into two categories: the bottom-up approach and the top-down approach. The bottom-up category models sensors and sensor networks from the device perspective. SensorML [5], IEEE1451 [6], and GSN [7] fall under the bottom-up category. SensorML models the sensor related processes for consistent handling. IEEE1451 defines transducer interface to access and manage smart transducers. GSN abstracts sensor data sources, specification, and query tools into XML description offering plug-and-play detection and deployment. The researches in bottom-up category noticed the heterogeneous and streaming characteristic; the other three were not in their consideration. The top-down category builds from the application perspective by analyzing functional requirements. The sensor network services platform (SNSP) [8], Cougar [9], sensor information networking architecture and application (SINA) [10], and COSMOS [11] belong to the top-down category. SNSP proposed a set of fundamental services and interfaces primitives to provide query operations on sensor networks. Cougar offered an in-network query processing mechanism. SINA attempt to find an optimal way to facilitate querying, monitoring, and tasking of sensor networks. COSMOS provided a method to integrate data over heterogeneous sensor networks and defined a standardized communication protocol and message formats. The researches in this category were all focusing on the heterogeneous characteristic; the other four were not considered in their papers.

Those methods mentioned above could not handle the sensor data according to all characteristics and support dynamic processing in an open-loop environment. Compared with those frameworks, the proposed SDP system which belongs to the bottom-up category pays attention to all the five characteristics and differs in the following aspects:

1. Based on unified data model, data compression method employing PSO algorithm is present to reduce redundancy.
2. The proposed system adopts data stream technology to monitor real-time and streaming exception situations.
3. Relational and NoSQL databases are compared and analyzed to seek a better model for massive sensor data storage.

3. Preliminary Notions

3.1. Particle Swarm Optimization. Particle Swarm Optimization (PSO) [12–14] proposed by Kennedy is an artificial intelligent algorithm aiming to help finding the optimal state in D-dimensional search space with the help of the interactions between particles in the swarm.

Every individual particle moves toward the position and position. The position is the best position found by particle so far. The position is the best position found by the swarm so far. Particle moves itself according to its velocity and current position. The velocity and position of particles can be updated by the following equations:

\[
\begin{align*}
\vec{v}_i &\leftarrow w \cdot \vec{v}_i + c_1 \cdot \text{rand}_1 \cdot (\vec{p}_{best} - \vec{x}_i) + c_2 \cdot \text{rand}_2 \cdot (\vec{g}_{best} - \vec{x}_i) \\
\vec{x}_i &\leftarrow \vec{x}_i + \vec{v}_i,
\end{align*}
\]

Here, refers to the velocity of particle , and is the position of the particle . is the personal best position of particle and is the global best position of the swarm. The inertia weight is used to control exploration and exploitation [15]. The particles maintain high velocities with a larger and low velocities with a smaller . A larger can prevent particles from becoming trapped in local optima, and a smaller encourages particles exploiting the same search space area. The constants and are used to decide whether particles prefer moving toward a position or position. and are random variables between 0 and 1.

The structure of PSO is shown in Algorithm 1. The after the iterations is the optimal position we seek.

3.2. Data Compression Measurement Indices. To evaluate the accuracy and the compression rate of data, we employ Amplitude Error index to represent accuracy and Compression Ratio index to stand for compression rate.
Begin
  Initialize a population of particles with random positions and velocities on $D$ dimensions in the search space.
  For each iteration $k$:
    begin
      For each particle $i$ in the swarm:
        Begin
          (1) Calculate new $\vec{V}_i$ using (1).
          (2) Update the position $\vec{X}_i$ according to (2).
          (3) Evaluate the value $f_{V_i}$ of fitness function $f$ with current position $\vec{X}_i$, and update $\vec{p}_{best}_i$ to $\vec{X}_i$.
            if $f_{V_i}$ is better than the fitness of $\vec{p}_{best}_i$.
        end
      end
    end
  end

Algorithm 1

3.2.1. Amplitude Error Index. Amplitude Error (AE) [16] is a measure of similarity between the input numerical list and the output numerical list of a method. It is often used to evaluate the accuracy of ECG data compression [17]. The smaller AE value between the recovered list from the output list of a compression method and the original ECG list reflects that the higher accuracy is achieved by this method. Our sensor data compression is similar to ECG data compression; we borrow this index to evaluate our compression accuracy.

Given an input numeric list, $\text{IList} = \{u_1, u_2, \ldots, u_n\}$, where $u_i$ is the $i_{th}$ number in $\text{IList}$, and a numeric list, $\text{RList} = \{V_1, V_2, \ldots, V_n\}$, where $V_i$ is the recovered value from the compressed data, AE is expressed by

$$AE = \frac{\sum_{i=0}^{n} (u_i - V_i)^2}{n}. \quad (3)$$

3.2.2. Compression Ratio Index. Compression Ratio (CR) [18] also comes from ECG data compression. It is a measure of the changes in the data amount. The smaller CR value represents that the larger amount of data have been compressed.

Given an original numeric list, $\text{OList} = \{u_1, u_2, \ldots, u_n\}$, where $u_i$ is the $i_{th}$ number in $\text{OList}$, and a compressed numeric list, $\text{CList} = \{V_1, V_2, \ldots, V_m\}$, where $V_i$ is the remaining value, and $m \leq n$, CR is expressed by

$$CR = \frac{m}{n}. \quad (4)$$

3.3. Fundamental Elements in the IoT. In an IoT information system, there are four fundamental concepts [19–21].

Entities. These include all RFID-tagged entities, such as items, cases, pallets, and even patients with RFID [22] bracelets.

Readers. There are two types of readers. One type is RFID reader; the other type is sensor reader which refers to the base station of WSNs. RFID readers use radio-frequency signals to communicate with RFID tags and also create business events, which describe the life cycle of an entity. Base stations collect data from the gateways of WSNs and stream all the data out to the Internet.

WSNs. WSNs collect and aggregate the environment information through sensors and communicate with base stations through gateways.

Container. A container is symbolized to represent where entities or readers or sensors locate. It may be a warehouse or a truck. The common identification of a container is the Global Location Number (GLN) [23] code defined by Globe Standard 1 (GS1) [24]. In general, more than one reader and at least one sensor are deployed in one container for tracking entities and monitoring environment.

4. Sensor Data Processing System

Figure 1 shows the components of proposed IoT sensor data processing (SDP) system and its relationship with the applications in the open-loop environment. We treat each piece of sensor data as an event, called sensor event. In the open-loop environment, there are WSNs deployed in physical containers and applications requiring information exchange. The SDP system consists of Observation Capturer, Real-Time Exception Monitor, and Dynamic Compression, three components. Sensor data streamed into SDP are in unified format, called Observation. One single index which describes a physical phenomenon state is called Phenomenon Event in SDP. Observation Capturer receives all the Observations from distributed base stations and split Observations into different Phenomenon Event according to the type of index. Real-Time Exception Monitor detects exception Phenomenon Event and pushes exception warning out. Dynamic Compression employs PSO algorithm to train threshold to compress Phenomenon Event.

4.1. Unified Sensor Data Definitions. We first introduce the new concept of Observation (O for abbreviation), which is a data object recording the basic information a sensor
and the values of phenomena. A phenomenon is a collected condition state. For example, a temperature of value 23.2°C is a phenomenon.

**Definition 1** (Observation). Consider

\[
O := <S_{addr}, \text{Time}, P>
\]

\[
P := \{<\text{ph}_\text{type}, \text{ph}_\text{value}, \text{ph}_\text{unit}>^+\}.
\]

Here, \(S_{addr}\) refers to the address of the sensor. The addresses of sensors in different WSNs differ with each other. The most common types of \(S_{addr}\) are \(<\text{gateway ip}, \text{sensor_set_off}>\), and \(\text{sensor_ip}\). Time means the collected time of an Observation. \(<\text{ph}_\text{type}, \text{ph}_\text{value}, \text{ph}_\text{unit}>\) is a triple, where \(\text{ph}_\text{type}\) refers to the type of a phenomenon, \(\text{ph}_\text{value}\) refers to the value of a phenomenon, and \(\text{ph}_\text{unit}\) denotes the unit of \(\text{ph}_\text{value}\). The sign + indicates that there may be more than one triple of phenomena generated by a sensor.

The following is an example of \(O\).

**Examples**

\[
<"192.168.0.12", "01", 2014-04-30 14:52:36, \{"air\_temperature", 23.3, "C"}, <"air\_humidity", 65.0, "%rh"}>
\]

We split Observations into Phenomenon Event (PE for abbreviation) list.

**Definition 2** (Phenomenon Event). Consider

\[
PE := <\text{Ph}_\text{type}, S_{addr}, \text{Time}, \text{Value}, \text{Unit}>.
\]

Here, \(\text{Ph}_\text{type}\) refers to the type of a phenomenon. \(S_{addr}\) and Time are inherited from Observation. Value is the numeric value of a phenomenon recorded by a sensor. Unit denotes the unit of Value.

The example in Definition 1 is split into two PEs:

\[
\begin{align*}
PE_1 &= <"air\_temperature", "192.168.0.12\_01", 23.3, "C", 2014-04-30 14:52:36> \\
PE_2 &= <"air\_humidity", "192.168.0.12\_01", 65.0, "%rh", 2014-04-30 14:52:36>.
\end{align*}
\]

**4.2. Collection**

**4.2.1. Observation Capture.** Base stations which wear the managing gateway of WSNs parse pieces of raw sensor data into the form of Os. Once Observation Capturer catches an \(O\), it splits \(O\) into several PEs and puts them into time windows according to their \(\text{Ph}_\text{Type}\) and \(\text{S}_{addr}\) attributes. A time window, which is set to output a list when time is up or the list is full, collects PEs for a preset time. Phenomena with the same \(\text{Ph}_\text{Type}\) and \(\text{S}_{addr}\) are pushed into the same time window. When a time window is full or the time is up, Observation Capturer gathers the PEs in the window into a PE list and passes this list to Dynamic Compression. The PEs in this list contain values of the same phenomenon and from the same sensor. So, the values are comparable.

**4.2.2. Dynamic Compression.** Dynamic Compression (DC) component, here, tries to compress data discarding the redundant PEs to reduce the load of Real-Time Exception Monitor system and data repository.

One data compression method is to record the first value in a sequence which contains the continuous same values. Hence, once a new value comes into the sequence, it will be the next recorded value. Using this method, PEs generated by a sensor containing two different air temperature values, 23.3°C and 23.4°C, would be two recorded Phenomenon Events. However, 0.1°C, the difference between 23.3°C and 23.4°C, does not mean a significant change of environment in some situations, such as in the greenhouse.

The other way called Threshold Compression is to preset a threshold and then discard the values whose difference with the last record is less than the threshold. This is the most common data compression method. It has three limits when it comes across the IoT sensor data. Firstly, the bigger the threshold is, the more values would be thrown away causing lower accuracy. In other words, when the threshold is reduced, higher accuracy will be achieved but more redundant data are stored. The previous method is one special case of this method with the smallest threshold zero. So, finding an appropriate threshold is a balance between accuracy and storage size. Secondly, there are various types of phenomena in the IoT. Presetting thresholds for all phenomena needs a lot of work. Thirdly, sensor data change all the time. A current appropriate threshold will be improper for future data sequence.

Here, DC seeks a threshold for each phenomenon type dynamically to balance accuracy and compression rate with the help of PSO algorithm and the crossbreeding of particle swarms. The primary goal of DC is to get the minimum AE. Based on this goal, we try to reduce the redundancy of PEs in PE list making smaller CR.
Once a PEList comes, DC executes the following steps to get a compressed PE list, CPEList.

**Step 1.** Initialize two particle populations of size m with random positions and velocities on \( [x_{\text{Min}}, x_{\text{Max}}] \), where \( x_{\text{Min}} \) represents the minimum value of the searching space, and \( x_{\text{Max}} \) refers to the maximum value of searching space. One population, \( \text{CRP} = \{ \text{CRP}_1, \text{CRP}_2, \ldots, \text{CRP}_m \} \), adopting CR as fitness function is used for finding the threshold in \( D \) to get minimum CR. The other population called \( \text{AEP} = \{ \text{AEP}_1, \text{AEP}_2, \ldots, \text{AEP}_m \} \), employing AE as fitness function is assigned to search the threshold in \( D \) to achieve minimum AE.

**Step 2.** Update the velocity and position of all particles in CRP and AEP with (1) and (2).

**Step 3.** For each \( i \in (1, \ldots, m) \), let \( \text{CRP}_i \) and \( \text{AEP}_i \) be parents, breed two children, \( n_{\text{CRP}} \) and \( n_{\text{AEP}} \), and calculate their positions with

\[
    x_{n_{\text{CRP}}} = p \cdot x_{\text{CRP}} + (1.0 - p) \cdot x_{\text{AEP}},
\]

\[
    x_{n_{\text{AEP}}} = p \cdot x_{\text{AEP}} + (1.0 - p) \cdot x_{\text{CRP}},
\]

where \( p \) is a random value between 0 and 1.

**Step 4.** Replace \( \text{CRP}_i \) with \( n_{\text{CRP}} \) and \( \text{AEP}_i \) with \( n_{\text{AEP}} \).

**Step 5.** Generate compressed PE lists for all particles in CRP with their positions as thresholds. Evaluate CRs for the particles in CRP using their compressed PEList with (4).

**Step 6.** Generate compressed PE lists for all particles in AEP with their positions as thresholds. For each particle, reconstruct time-value line with the PEs in its CPEList using the process described in Section 4.2.3, and calculate AE between PEList and CPEList.

**Step 7.** For each particle in CRP, compare its current CR value with the CR of its \( p_{\text{best}} \) position. If the current value is less, then update \( p_{\text{best}} \) and its current position.

**Step 8.** For each particle in AEP, compare its current AE value with the AE of its \( p_{\text{best}} \) position. If the current value is less, then update \( p_{\text{best}} \) and its current position.

**Step 9.** Determine the global best particle, \( \text{gbCRP} \), in CRP with the smallest CR value.

**Step 10.** Determine the global best particle, \( \text{gbAEP} \), in AEP with the smallest AE value.

**Step 11.** Use the position of \( \text{gbCRP} \) as threshold to generate a compressed list, CRCPEList from PEList, and calculate the AE value between CRCPEList and PEList.

**Step 12.** Compare the AE value from Step 11 to the AE value of \( \text{gbAEP} \). If the AE value of \( \text{gbAEP} \) is bigger, then set the position of \( \text{gbAEP} \) to \( \text{gbCRP} \)’s position.

**Step 13.** Repeat Step 2 to Step 12 until a sufficiently good AE or a maximum number of iterations are met.

The position of \( \text{gbAEP} \) after the above 13 steps is the optimal threshold that can help us to compress PEList into CPEList balancing accuracy and compression rate.

### 4.2.3. Time-Value Reconstruction

All of the PEs in a PE list are of the same type; two key attributes in a PE are time and value. So reconstruction of a PE is to recalculate the phenomenon value with a time. The time-value pairs in the compressed PE list are discrete points. For reconstruction, we make Time attribute as \( X \) coordinate and Value attribute as \( Y \) coordinate. For all PEs in a compressed PE list, mark \((\text{Time}, \text{Value})\) points one by one, and draw lines between two adjacent points. The \( Y \) values in the connected line are the reconstructed values of PEs for a certain time.

For example, given a compressed Phenomenon Event List, CPEList,

\[
    \text{CPEList} = \{ <\text{air\_temperature}, \text{192.168.0.12}, \text{01}, \text{25.8}, \text{C}, \text{2013-06-19 21:03:00}>, <\text{air\_temperature}, \text{192.168.0.12}, \text{01}, \text{25.6}, \text{C}, \text{2013-06-19 21:34:40}>, <\text{air\_temperature}, \text{192.168.0.12}, \text{01}, \text{25.4}, \text{C}, \text{2013-06-19 22:09:05}>, <\text{air\_temperature}, \text{192.168.0.12}, \text{01}, \text{25.2}, \text{C}, \text{2013-06-19 22:42:41}>, <\text{air\_temperature}, \text{192.168.0.12}, \text{01}, \text{25.0}, \text{C}, \text{2013-06-19 23:08:40}> \}
\]

The points on the line in Figure 2 are the reconstructed values of PEs for a certain time.

#### 4.3. Real-Time Exception Monitor

As explained, sensor data reflects the phenomenon status in the physical space. Hence,
sensor data are closely associated with the containers. The monitored containers require that the phenomenon states are stable in certain intervals. For example, a warehouse in a cold train loaded with meats requires air temperature maintained at \([-18^\circ C, 0^\circ C]\). Once the air temperature is out of this interval, especially, beyond the limits, warnings are required to be pushed out to the applications.

Real-Time Exception Monitor (REM) receives compressed Phenomenon Event List, CPEList, from Dynamic Compression (DC) to detect exception situations. Although the elements in CPEList are not the complete captured Phenomenon Events, they are still key elements collected with a threshold. Therefore, the Phenomenon Events in CPEList can represent the actual exception situations.

REM maintains a Rule Repository and a core execution component Real-Time Exception Monitor Core (REMC for short). Rule Repository stores a set of rules defining the standard phenomenon interval. Every rule is formatted as a RL:

\[
RL := \langle GLN, Ph\_Type, Min, Max, Unit \rangle.
\]

Here, GLN is a thirteen-digit number used to identify parties and physical locations. Ph\_Type denotes the type of a phenomenon. Min and Max describe a standard range with Unit of Ph\_Type. The following example shows a standard air temperature interval \([-18^\circ C, 0^\circ C]\) of a warehouse with GLN 6901404000029 in meat cold chain.

**Examples**

\[
<6901404000029, "air\_temperature", -18, 0, "C">.
\]

REMC focuses on the exception monitoring execution. Once a compressed PE list comes, REMC first uses the common sensor address of elements in CPEList to get the respective GLN number. Then, REMC retrieves RL for the container from Rule Repository with the GLN and the common Ph\_Type of PEs. From all hit RLs, REMC looks for the maximum Min and minimum Max to form the minimum standard range for all types of entities. For each PE in the compressed PE list, if the value is not in the range, REMC will send out a warning.

4.4. Sensor Data Storage. As described in Section 3, an information system manages several containers and at least one sensor is deployed in a container. Meanwhile, the sensors collect information of various phenomena in seconds. As a result, the number of Phenomenon Events generated each day is large. Although PEs have been compressed in Dynamic Compression, the stored number still increases fast. To find a better way to store and retrieve Phenomenon Events efficiently supporting environment information discovery for entities, we attempt to maintain sensor data in two ways, SQL database and NoSQL database.

4.4.1. SQL Database. The structure of sensor data in SQL database is designed as Figure 3. A container contains one or more sensors, and sensors generate several types of events. Considering the insertion and selection operations focus on single type of phenomenon, we classify events into different tables according to their types to reduce the size of Phenomenon Event table. In this way, if more phenomena are monitored, more tables will be created. To improve data retrieving efficiency and reduce the storage space, we define fields using variable characters instead of characters, float replacing decimal number, and timestamp rather than datetime.

**Insertion.** PEs in the compressed PE list are stored directly into corresponding table grouped by their Ph\_Type.

**Query.** The most common query on sensor data like the following example takes a GLN number and a time slot to search the records of a specific phenomenon. GLNCode in Container table and time in Phenomenon Event tables appear often:

\[
\text{Select } * \text{ from AirTemperatureEvent a join sensors on } a.SensorID = s.SensorID \text{ join Container c on } s.ContainerID = c.ContainerID \text{ where c.GLNCode = '1' and a.time between '2013-06-19 20:00:00' AND '2013-06-20 16:59:59'}.
\]

For decreasing the event retrieving time, we add indices on GLNCode field and time field.

4.4.2. NoSQL Database. NoSQL database maintains sensor-GLN collection and Phenomenon Event collection. Algorithm 2 shows the storage example tuple in NoSQL database. Since NoSQL database stores data in collections, this paper divides the Phenomenon Event and basic information of sensor and containers into two collections. Sensor-GLN collection maps the relationship between sensor address and GLN number of physical container managing [S\_addr, GLN] tuples. Phenomenon Event collection stores [Ph\_Type, S\_addr, Ph\_Value, Ph\_Unit, Time] tuples. S\_addr, which represents the sensor deployed in container and the data source of Phenomenon Events, is the key value to build relationships between physical containers and the actual values of phenomena, while physical containers are identified with GLN code.

| Example of Sensor-GLN collection |
|----------------------------------|
| "Sensor Addr": "222.73.86.252.0003", |
| "GLN": "6901125000107"; |

| Example of Phenomenon Event collection |
|----------------------------------------|
| "Sensor_Addr": "222.73.86.252.0003", |
| "Ph\_Type": "air\_temperature", |
| "Ph\_Value": 23.6, |
| "Ph\_Unit": "C" |
| "Time": 2015-02-03, |

**Algorithm 2:** Example tuple in NoSQL database.
Insertion. PEs in the compressed PE list are stored directly into Phenomenon Event collection without extra operations, since all tuples in this collection record their Ph_Type.

Query. The most common query is also finding the Phenomenon Event that happened in a container during a time slot. This query would require executing querying operation two times. Firstly, get the S_addr list with GLN from sensor-GLN collection. Secondly, for each S_addr, find all events records generated over this period.

To reduce the query times, there is another way to store PEs. It is to find the GLN number with S_addr before inserting a PE and store [Ph_Type, GLN, Ph_Value, Ph_Unit, Time] tuples in Phenomenon Event collection. However, as to the current situation, insertion operations of sensor data happen more frequently than queries. Hence, we adopt the first design on NoSQL DB.

Table 1: Parameters of PSO.

| $\omega$ | $c_1$ | $c_2$ | $x_{\text{Min}}$ | $x_{\text{Max}}$ | $v_{\text{Max}}$ |
|----------|-------|-------|-----------------|-----------------|-----------------|
| 0.4      | 1.8   | 1.8   | 0.0             | 0.5             | 0.2             |

5. Evaluation

5.1. Experiment Setup. We implemented a prototype of the proposed sensor data processing system SPS. Our experimental platform consists of a PC running Windows 7 professional with 4.00 GB memory and Intel(R) Core(TM) i3-3220 CPU @ 3.30 GHz processor. The parameters of PSO are set as in Table 1. The selected SQL database is MySQL, and the representative of NoSQL database is MongoDB.

5.2. Performance Evaluation. In order to evaluate performance of our SDP system, three experiments are conducted.
The first experiment is to show whether DC can compress data dynamically according to the characteristics of data sequences. The second experiment is to show how the number of PEs affects the compression speed and how the selections of database affect the storage speed. The third experiment is to show how the setting of database affects the tracing speed.

Testing data came from Wuxi Institute of Fudan University. These data were recorded every 5 seconds by the sensors in a greenhouse. For each experiment, we set the window size to 30 seconds. To test the performance of our framework, all PEs were submitted to the sliding window in 30 seconds.

There are many factors affecting the performance of the proposed IoT sensor data processing SDP system, such as the characteristics of the PEs, the number of PEs, and the settings of database. In our evaluation experiments, we focused on how the number of PEs and the selections of database affect performance. Meanwhile, we try to evaluate whether the data compression (DC) can compress data dynamically according to the characteristics of data sequences. The initial parameters of PSO are set as in Table 1 before experiment.

The first experiment was conducted with the 1867 pieces of air temperature records and 1868 pieces of pressure records as PEs. We measured the accuracy, AE, and the compression rate, CR, when DC outputs a compressed PE list for a phenomenon type. The experimental results are shown in Figures 4 and 5. We can see that DC keeps all PEs whose value of pressure differs from their previous recorded value, while DC finds the trend of air temperature in Figure 5 and records only 5 points. DC can find a threshold to balance the accuracy and compression rate. Meanwhile, the value of AE index approximates 0.00321, and the value of CR is near 0.5133 in Figure 4. And the data compression of air temperature in Figure 5 can achieve AE value of 1.28197 and CR value of 0.00268. This part of experiment shows that the proposed dynamic data compression can identify the trends of data and balance the accuracy and compression rate. Redundancy from sensor data can be avoided.

The second experiment was conducted with the number of sensor data varying from 500 to 1000000. We measured the time when DC begins compressing, the time of DC after compression, and the time after storage. The second one subtracted by the first one was considered as the compression time, and the last one subtracted by the second one was taken as the storage time. The result is shown in Figure 6 and it reflected that the compression time of DC has linear relationship with the number of PEs. The time storing PEs into MySQL database increases faster than the time storing data into MongoDB.

The third experiment was conducted using a query to trace the air temperature information during the lifecycle of an entity with an EPC. The number of recorded PEs in database varied from 69880 to 1985247. The query time of MySQL and MongoDB is shown in Figure 7. We learned that the tracing time of MongoDB grows faster than the time of MySQL. The reason is that MongoDB had to execute two queries when the number of tuples increases. However, comparing to the storage time, the time cost by the sensor data queries of MongoDB is short.

Hence, when applications are faced with a large amount of sensor data storage, it is better to store sensor data into nonrelational databases, while when the number of environment information discovery queries is larger than storage, rational databases have better performance.

6. Conclusion

In this paper, we present a new IoT sensor data processing (SDP) system to process sensor data dynamically in the context of the Internet of Things. First, heterogeneous sensor
data are captured and transformed into unified data format. Second, Particle Swarm Optimization algorithm is employed to do data compression avoiding redundancy and helping to reduce the load of database by adding crossbreeding operation on PSO algorithm. The proposed SDP system detects exception situations by setting the standard phenomenon rules of containers. Meanwhile, an appropriate type of database suitable for the sensor data storage in the IoT is sought and analyzed in this paper. The experimental results show that the proposed compression method can find a threshold achieving high compression rate and keeping accuracy, and NoSQL database has better performance in sensor data storage while relational database does better when executing environment information discovery queries.

**Conflict of Interests**

The authors declare that there is no conflict of interests regarding the publication of this paper.

**References**

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] M. C. Domingo, "An overview of the internet of things for people with disabilities," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 584–596, 2012.

[3] L. Zheng, H. Zhang, W. Han et al., "Technologies, applications, and governance in the internet of things," in *Internet of Things—Global Technological and Societal Trends from Smart Environments and Spaces to Green ICT*, River Publishers, 2011.

[4] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[5] M. Botts and A. Robin, *Sensor Model Language (SensorML) Implementation Specification, OpenGIS Implementation Specification*, 2007.

[6] M. Botts and A. Robin, "Sensor Model Language (SensorML) Implementation Specification, OpenGIS Implementation pification, 2007," *IEEE Instrumentation and Measurement Society, IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats*, IEEE Std 1451.0, 2007.

[7] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *Proceedings of the 8th International Conference on Mobile Data Management (MDM '07)*, pp. 198–205, May 2007.

[8] M. Sgroi, A. Wolisz, A. Sangiovanni-Vincentelli, and J. M. Rabae, "A service-based universal application interface for ad hoc wireless sensor and actuator networks," in *Ambient Intelligence*, pp. 149–172, Springer, Berlin, Germany, 2005.

[9] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM Sigmod Record*, vol. 31, no. 3, pp. 9–18, 2002.

[10] C.-C. Shen, C. Srisathapornphat, and C. Jaiakao, "Sensor information networking architecture and applications," *IEEE Personal Communications*, vol. 8, no. 4, pp. 52–59, 2001.

[11] M. Kim, J. W. Lee, Y. J. Lee, and J.-C. Ryou, "Cosmos: a middleware for integrated data processing over heterogeneous sensor networks," *ETRI Journal*, vol. 30, no. 5, pp. 696–706, 2008.

[12] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[13] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*, pp. 760–766, Springer, New York, NY, USA, 2010.

[14] F. Van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[15] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," in *Proceedings of the Congress on Evolutionary Computation*, vol. 1, pp. 101–106, IEEE, Seoul, Republic of Korea, May 2001.

[16] L. Gang, F. Jing, L. Ling, and Y. Qilian, "Fast realization of the LADT ECG data compression method," *IEEE Engineering in Medicine and Biology Magazine*, vol. 13, no. 2, pp. 255–258, 1994.

[17] S. M. S. Jalaleddine, C. G. Hutchens, R. D. Strattan, and W. A. Coberly, "ECG data compression techniques—a unified approach," *IEEE Transactions on Biomedical Engineering*, vol. 37, no. 4, pp. 329–343, 1990.

[18] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolic, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 604–611, IEEE Computer Society, Paris, France, 2002.

[19] E. Welbourne, L. Battle, G. Cole et al., "Building the internet of things using RFID: the RFID ecosystem experience," *IEEE Internet Computing*, vol. 13, no. 3, pp. 48–55, 2009.

[20] I. Granbæk, "Architecture for the Internet of Things (IoT): API and interconnect," in *Proceedings of the 2nd International Conference on Sensor Technologies and Applications (SENSORCOMM ’08)*, pp. 802–807, Cap Estrel, France, August 2008.

[21] R. Want, "An introduction to RFID technology," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 25–33, 2006.
[23] Global Location Numbers (GLN), GS1, http://www.gs1.org/docs/idkeys/GS1_Global_Location_Numbers.pdf.

[24] GS1, http://www.gs1.org/.