International Journal of Grid and Distributed Computing
Vol. 11, No. 1 (2018), pp.27-36
http://dx.doi.org/10.14257/ijgdc.2018.11.1.03

Reliability Logic Model for Incomplete Information in Sensor Monitoring Databases

Siwoo Byun[1*] and Seok-Woo Jang[2]

[1,2]Dept. of Software, Anyang University
708-113, Manan-gu, Anyang-shi, Kyonggi-do 430-714, South Korea
[1]swbyun@anayng.ac.kr, [2]swjang@anayng.ac.kr

Abstract

Ubiquitous network consists of a large number of sensor nodes that combine physical sensing capabilities with networking capabilities. The sensors send data to a central monitoring database server where the data is aggregated.

The objective of this research is to reduce the uncertainty of user query for incomplete information in the sensor databases. Typical method to give more informative answers to users is interpolation method. And there has been few researches in the area of handling uncertainty for incomplete information without user-defined probability. But uncertainty handling mechanism with user-defined probability approaches is not desirable to let users catch the meanings of the stored data and the users are not familiar to difficult and complex probability functions.

In this paper, a new approach of handling uncertainty, Reliability Logic(RL) model, is proposed, which provides users generality and naturalness of query expression for incomplete monitoring databases. In our RL model, two basic reliability operators are introduced into restriction clause. The extension to RL model can give the users expressive power of monitoring query, notify the degree of uncertainty and reduce the uncertainty of the user query.

Keywords*: Reliability Logic, Sensor data, Algorithm, Uncertainty monitoring*

1. Introduction

1.1. Sensor Network

The widespread deployment of sensors is transforming the physical world into a computing platform. Modern sensors not only respond to physical signals to produce data, they also embed computing and communication capabilities. They are thus able to store, process locally and transfer the data they produce.

Sensor data is the output of a device that detects and responds to some type of input from the physical environment. The output may be used to provide information or input to another system. Commonly monitored parameters include temperature, humidity, pressure, wind direction and speed, illumination intensity, vibration intensity, sound intensity, power-line voltage, chemical concentrations, pollutant levels and vital body functions.

Sensor database is in integral component of the increasing reality of the *Internet of Things* (IoT) environment. Much of the data transmitted is small sensor data. The huge volume of data produced and transmitted frequently from sensing devices can provide a lot of information but is often considered the next big data challenge for smart IoT businesses. Recently, various sensor database systems, including TinyDB and Cougar,

Received (August 26, 2017), Review Result (December 19, 2017), Accepted (January 12, 2018)

ISSN: 2005-4262 IJGDC

have been developed for the efficient management of sensor data in the ubiquitous environment [1].

IoT server monitors the physical world by querying and analyzing sensor data. Examples of monitoring applications include supervising items in a factory warehouse, gathering information in a disaster area, or organizing vehicle traffic in a large city. Typically, these applications involve a combination of stored data (a list of sensors and their related attributes, such as their location) and sensor data. We call *sensor database* the combination of stored data and sensor data [2].

1.2. Temporal Database for Monitoring Sensor Nodes

Conventional databases are designed to capture the most recent data, that is, current data. As new data values become available through updates, the existing data values are removed from the database. Such databases mainly capture a snapshot of reality. Although conventional databases serve some applications adequately well in the sense of performance, they are insufficient for those in which past and/or future data are required, due to the overhead of their own programming in managing temporal data. What is needed is a database that fully supports the storage and querying of information that varies over time. In the broadest sense, a database that maintains past, present, and future data is called a *temporal database*[3]

In temporal databases, the likelihood of missing information increases because of the vast amount of information being stored. Furthermore, users may want to maintain only selective portions of history. Hence, a mechanism must be provided to store and query incomplete temporal information. There are two mechanisms that must be considered in the data models for incomplete information in temporal databases: *estimation* mechanism and *uncertainty handling* mechanism associated with the estimation.

First, for a given object, we may know the values for a given attribute at some points in time but the values at other points in time may be unknown. In this case, conventional databases mark it as a special symbol called a *null* value. But, more powerful mechanism must be provided to give more informative answers to users, for instance, *interpolation* method.

Second, for a given object, we may estimate missing values for a given attribute at some points in time by known methods. But, estimation of missing values has uncertainty because they are derived values not stored values which can be evaluated to *true* or *false* without uncertainty. In this case, missing values which have uncertainty must be handled differently from stored values. If only estimation mechanism is provided to users without uncertainty handling mechanism, users must treat estimated data as 100 percent correct data without filtering their uncertainty and are not able to specify the acceptable range of the estimation error. What is needed is an uncertainty handling scheme to overcome this shortcoming.

It is easy to handle uncertainty for stored data, *i.e.*, complete information, supplied with user-defined probability, for instance, (pink=0.4, green=0.6) in color attribute. However, user-defined probability approaches require additional space to preserve their probabilities in databases. Furthermore, it is not desirable to let users catch the meanings of the stored data, characteristics of the estimation function, and their errors, and input manually their probabilities, since users are not familiar with difficult and complex probability functions.

2. Motivation

2.1. Monitoring Databases for Ubiquitous Sensor Network

With the development of sensor technologies for sensing various types of data and with advances in wireless communication technologies, there is increasing interest in, and research on, the application of technologies related to *ubiquitous sensor networks* (USNs)

in areas such as ecosystem monitoring, home automation, and car theft detection. A USN is a communication network in which various types of sensor nodes interconnected by means of wireless communication schemes [1].

The example of tiny sensor node is illustrated in Figure 1. The minimal hardware of this sensor node based on TinyOS is consist of 4Mhz, 8 bit RISC, 40k bit Radio, 4 K RAM, 128 K Program Flash, 512 K Data Flash, and AA battery pack. The example of fire monitoring application using the sensor network is also illustrated in Figure 2.

Figure 1. Example of Tiny Sensor Node

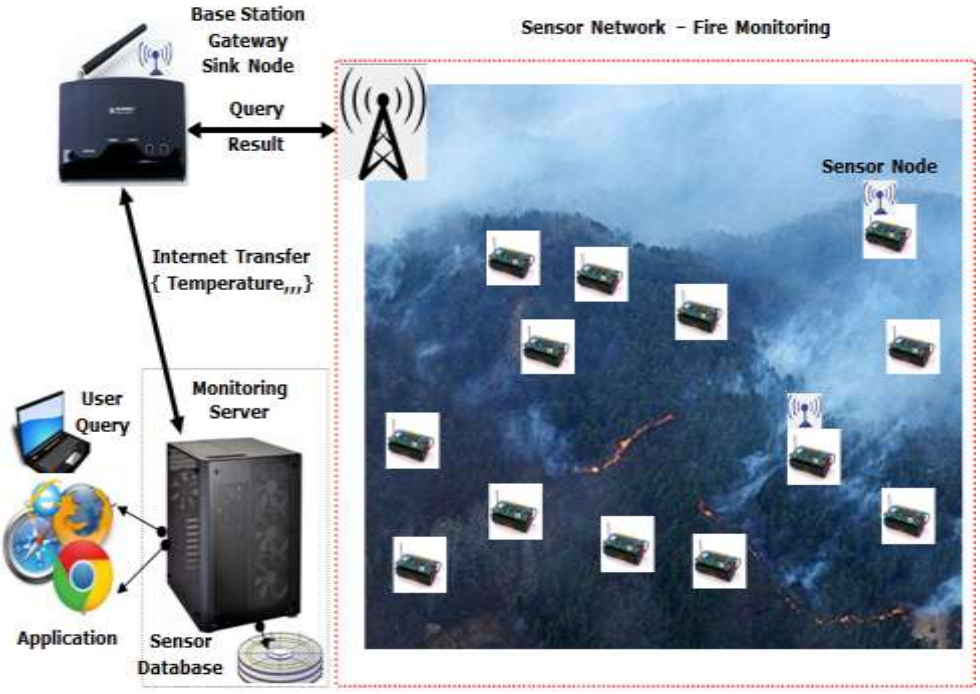

Figure 2. Architecture of Sensor Network (Fire Monitoring)

A sensor network consists of a large number of sensor nodes. Individual sensor nodes are connected to other nodes in their vicinity through a wireless network, and they use a multihop routing protocol to communicate with nodes that are spatially distant. A special nodes called *gateway* are connected to components outside of the sensor network through long-range communication (such as cables or satellite links), and all communication with users of the sensor network goes through the gateway node or base station.

USN applications can monitor the physical world by remote measuring and analyzing the sensor data. The sensor data table is unbounded, and sensor database management

system basically uses SQL-like query interfaces in the form of SELECT-FROM-WHERE[4].

A sensor node consists of sensing, processing, storage, and communication modules. However, it has following physical resource constraints.[5]

• **Communication**: The bandwidth of wireless links connecting sensor nodes is usually limited, on the order of a few hundred Kbps. In addition, the wireless network connecting the sensor nodes provides usually only very limited quality of service, has latency of high variance, and drops packets frequently.

• **Power consumption**: Sensor nodes have limited supply of energy, and energy consumption is one of the main system design considerations(Table 1). Small batteries provide about 3000mAh of capacity, powering the sensor node for approximately one year in the idle state and for one week under full load. Note that future sensor nodes will have sophisticated power management features; current smart sensor nodes already have three different sleep modes with several orders of magnitude different sensor power usages.

Table 1. Power Consumption of Sensor Operations

Sensor Node Operations	nAh
Transmitting a packet	20.0
Receiving a packet	8.00
Operating sensor for 1 sample	1.08
Accessing flash memory data	5.51
Reading a Sample from ADC	0.01

• **Computation**: Sensor nodes have limited computing power and memory sizes that restrict the types of data processing algorithms that can be deployed and intermediate results that can be stored on the sensor nodes.

• **Uncertainty in sensor readings**: Signals detected at physical sensors have uncertainty due to limitations of the sensor, and they may contain environmental noise. Sensor malfunctions might generate inaccurate data, and unfortunate sensor placement (such as a temperature sensor directly next to the air conditioner) might bias individual readings.

That is, it has limitations with regard to its capacity to process sensed data, the space available to store sensed data, and the amount of electric power available. This limitation leads to incomplete collection of sensor data in monitoring databases. Therefore, the monitoring system, which is based on the sensor data, must consider this unreliability and sufficiently notify the degree of reliability to users.

2.2. Limitation of Conventional Database Models

At first, we introduce uncertainty factor into a new SQL-like query which can reduce uncertainty of user query for incomplete information. We then attempt to notify the degree of uncertainty of the query to users.

It is convenient to define some terms related with temporal database concepts[6] before our discussion about handling incomplete information in temporal databases.

Definition 1 (instant) An *instant* is a time point on an underlying time axis. ■

Intuitively, the instants in a discrete model of time are isomorphic to the natural numbers, *i.e.*, there is the notion that every instant has a unique successor. So, time is linearly ordered; for the two unequal time points t and t', ether t is before t' or t' is before t on a time line. We will interpret time as a set of equally spaced and ordered time points

and denote it by T.T = {0, 1, 2, ...**now**..}. The symbol 0 is the relative beginning, and **now** is a special constant to represent current time. The value of **now** advances as the clock ticks. Any point beyond **now** is future time. In Figure 3, t0, t1, **now** are linearly ordered time points on a time line. For instance, t1 is an instant which is after t0 and before **now** on the time line.

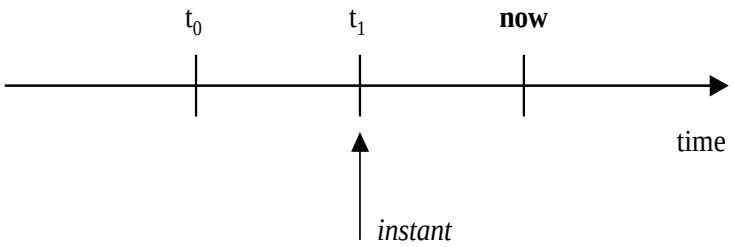

Figure 3. An Instant on a Time Line

Definition 2 (chronon) In a data model, a one-dimensional *chronon* is a non-decomposable time interval of some fixed, minimal duration. ■

Data models may represent a time line by a sequence of non-decomposable, consecutive time intervals of identical duration. these intervals are termed *chronons*. In Figure 4, t0, t1, t2 are linearly ordered time points on the time line. Between two consecutive time points, for instance, between t0 and t1, there is a chronon.

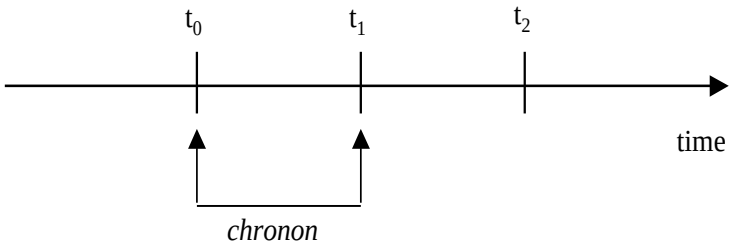

Figure 4. A Chronon on a Time Line

Definition 3 (temporal interpolation) The derivation of the value of a history at a chronon for which a value is not explicitly stored into the database, is referred to as *temporal interpolation.* This derivation is typically expressed as a function of preceding and/or succeeding(in time) values of the history.

Definition 4 (time-invariant attribute) A *time-invariant* attribute is an attribute whose value is constrained to not change over time. In functional terms, it is a constant-valued function over time. For instance, the name of an employee is a time-invariant attribute. ■

Definition 5 (time-varying attribute) A *time-varying attribute* is an attribute whose value is not constrained to be constant over time. In other words, it may or may not change over time. For instance, the salary of an employee is a time-varying attribute. ■

For the conventional relational model, unknown values are marked by a special symbol called a *null* value. Since some attribute values are unknown, a selection expression does not always evaluate to *true* or *false* for a given tuple. However, the missing values can be calculated from surrounding samples using known methods by interpolation functions. For example, in the *embedded interpolation procedure*(EIP) model[7], interpolation procedures are incorporated into the query part. Therefore, users are able to query missing values in the original measurements with SQL-like syntax illustrated by Figure 5. Clause (1), (2), and (3) are extensions to the standard SQL[4]. Clause (1), the BY METHOD-

clause, specifies the method and the continuous attribute it is applied to. Clause (2), the ENTRY-clause, gives the values of the varying base attributes, *i.e.*, it specifies the instant where temporal interpolation should be done. Clause (3), the STEP-clause, gives the interval for an equidistant sequence of interpolation values.

SELECT <projection clause>
FROM <table> **BY METHOD** <method> (<continuous_attr>) (1)
ENTRY (<base_attr>=<value>) (2)
STEP (<distance>) (3)
[**WHERE** <restriction clause>];

Figure 5. Syntax of Query in EIP Model

An instance of query appears in Figure 6. Answer of this query is a set of (*time, temperature*) pair from *measured_data* table by 30-second interval between 07:00 and 09:00. The interpolation function, *weighted_mean*, calculates missing values of *temperature* from stored values. We assume that the interpolation function always calculates missing values in any chronon.

SELECT *time, temperature*
FROM *measured_data* **BY METHOD** *weighted_mean(temperature)*
ENTRY (*time* = 07:00)
STEP (30 seconds)
WHERE *time* <= 09:00;

Figure 6. A Monitoring Query Instance in EIP Model

It is assumed that in relational data models only stored data can be represented and their values are correct, *i.e*. all facts not implied by the stored data are taken to be wrong. Answers consist of sets or multisets(or bags) of tuples with values which were evaluated to *true* in queries within only specified domains, *i.e*. set of stored data.

But, in the case of interpolated values, we must handle them more carefully. All stored values without null are always evaluated to *true* or *false* in restriction clause, *i.e*., WHERE clause, in a query because we assume that they are right values with no doubts. However, all interpolated values have uncertainty because interpolation function is also a kind of estimation process. Therefore, the direct evaluation of restriction clause which contains uncertain interpolated values to *true* or *false* has uncertainty, so answers of query is not reliable. This unreliability is a serious problem in conventional interpolation models. This is illustrated by Example 1.

Example 1 (Unreliable Evaluation Problems in Interpolation Models)
Suppose a SQL-like statement whose restriction clause has an interpolation attribute. A SQL-like statement regarding a user query whose answer contains interpolated values appears in Figure 7. The statement is extended from Figure 6 by appending an interpolation constraint, 'temperature < 100F'.

SELECT *time, temperature*
FROM *measured_data* **BY METHOD** *weighted_mean(temperature)*
ENTRY(*time* = 07:00)
STEP(30 seconds)
WHERE *time* <= 09:00 AND *temperature* < 100F ;

Figure 5. A SQL-like Statement which has Interpolation Constraint

The values of the interpolation attribute, *temperature*, have uncertainty because they are derived by interpolation function, *weighted_mean*, which could bear *interpolation errors*. Therefore, the temporal constraint has uncertainty and therefore, the restriction clause is also unreliable. This unreliability influences to answer of this query. ■

In order to avoid this unreliability, users must exclude the interpolation constraint. This means that the expressive power of user query is limited in the sense that users are not able to specify the acceptable range of the interpolation error. When a number of interpolation attributes are required in a user query and their interrelationships are complex, the degree of unreliability could increases significantly. Furthermore, if users want to select tuples with reliability measure for interpolated values, for instance, 'interpolation error < 10%', there is no way in these conventional data models.

3. Reliability Logic Control for Sensor Monitoring Databases

To provide users generality and naturalness of query expressions for incomplete information, the extension of conventional database model is required in the two following aspects. A first aspect is to provide *estimation mechanism*, and second aspect is to provide *uncertainty handling mechanism* associated with the estimation. If only estimation mechanism is given, *unreliable evaluation problems* which was illustrated in the Example 1 would occur.

Our goal is to offer a richer descriptive query interface which can more accurately handle incomplete information by providing uncertainty handling mechanism which can reduce the degree of unreliability of the user query. This uncertainty handling mechanism should be automatically driven by system without user interaction at execution time in order to provide easy access to most general users who are not familiar with the complex estimation process and its error calculation process. Finally, as we will see later on, in our model which we call *Reliability Logic(RL) model*, the expressive power of user query is extended in the sense that users can filter uncertainty of their query by specifying the acceptable range of the interpolation error in query expressions and can get reliability of the answers at query evaluation time.

We introduce two basic *reliability operator* into restriction clause *i.e*., WHERE clause in a new SQL-like query in order to handle the uncertainty of user query. These operators are a kind of conversion functions from reliability logic to Boolean logic and defined in the two following definitions. We assume that the estimation function is already given and reliability values are generated from the estimation function.

Definition 6 (greater-than reliability operator; %>)
E1 **%>** E2 is *true* if reliability of expression E1 is greater than expression E2. Otherwise, *false*. ■

For instance, a '(*temperature*>100F) %> 85' predicate is true if the probability that *temperature* is greater than 100F is greater than 85%.

Definition 7 (less-than reliability operator; %<)
E1 **%<** E2 is *true* if reliability of expression E1 is less than expression E2.
Otherwise, *false*. ■

For instance, a '(*temperature*>100F) %< 90' predicate is *true* if the probability that *temperature* is greater than 100F is less than 90%.

A new SQL-like query is extended from *EIP* model by incorporation reliability operators into restriction clause. This is illustrated in Figure 8.


```
SELECT <projection clause>
FROM <table> BY METHOD <method>(<continuous_attr>)
                ENTRY (<base_attr>=<value>)
                STEP(<distance>)
[WHERE <restriction clause>];

<restriction clause>::= <expression> |
                <expression>  %> <expression> |
                <expression>  %< <expression>
```

Figure 8. Syntax of Query in RL Model

An instance of query appears in Figure 9. The *DECLARE* statement specifies the estimation function *e*() which uses two reference time instants and the reliability function *r*(). The two functions can be defined by DBA or users. Answer of the query is a set of (*time, temperature*) pair in which probability of *'temperature* > 100F' is greater than 85% from measured_data table by 30-second interval between 07:00 and 09:00. The interpolation function, *weighted_mean*, calculates missing values of *temperature* from stored values. We assume that the interpolation function always calculates missing values in any chronon.

```
DECLARE METHOD weighted_mean
        ARGUMENT time t
        ON TABLE measured_data
        REFERENCE time t-1, time t+1
        ESTIMATION temperature e()
        RELIABILITY real r();
SELECT time, temperature
FROM measured_data BY METHOD weighted_mean(temperature)
                ENTRY (time = 07:00)
                STEP (30 seconds)
WHERE time <= 09:00 AND (temperature < 100F) %> 85;
```

Figure 9. A Monitoring Query Instance in RL model

4. Conclusions and Future Work

A new data model for incomplete information called *Reliability Logic Model*(RL) is devised to reduce the uncertainty of user query for incomplete information in monitoring database environments. The traditional uncertainty handling mechanism with user-defined probability approaches is not desirable to let users catch the meanings of the stored data and the users are not familiar to difficult and complex probability functions.

In RL model, users can get generality and naturalness of monitoring query expression for incomplete sensor information. In order to allow the users to use the reliability operators in RL, we used extended SQL-like query of conventional databases. However, the extended query of RL model would require additional overheads in database systems. We intend to further analyze this and show detailed techniques for implementing the RL model in general databases.

Acknowledgments

This paper is a revised and expanded version of a paper entitled “A Study on Incomplete Information Management for Monitoring Databases” presented at GST2017 (Jeju, South Korea, Dec., 2017). This research was supported by Basic Science Research

Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A1A01057675).

References

[1] K. Dong-Oh, L. Lei, S. In-Su, K. Jeong-Joon and H. Ki-Joon, "Spatial TinyDB: A Spatial Sensor Database System for the USN", Int. Journal of Distributed Sensor Networks, **(2013)**, pp.1-10.
[2] P. Bonnet, J. Gehrke and P. Seshadri, "Towards Sensor Database Systems", Proceedings of the Second International Conference on Mobile Data Management, Hong Kong, **(2011)**, pp.3-14.
[3] C.J. Date, H. Darwen and N. Lorentzos, "Time and Relational Theory", Second Edition: Temporal Databases in the Relational Model and SQL, MK Publication Company, Preface, **(2014)**.
[4] SQL, https://wikipedia.org/wiki/SQL, **(2016**).
[5] C1, Yong Yao and J. Gehrke, "Cougar Approach to In-Network Query Processing in Sensor Networks", ACM SIGMOD Record, vol. 31, no. 3, **(2002)**, pp.9-18.
[6] S. Christian, "a Consensus Glossary of Temporal Database Concepts", ACM SIGMOD Record, vol. 23, no. 1, **(1994)**, pp. 52-64.
[7] N. Leonore, "Optimization and Evaluation of Database Queries Including Embedded Interpolation Procedures", ACM SIGMOD Record, vol. 20, no. 2, **(1991)**, pp. 118-127.

Authors

Siwoo Byun, he received his B.S. degree in Computer Science from Yonsei University in 1989 and earned his M.S. and Ph.D. in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1991 and 1999. Currently, he is teaching and researching in the area of distributed database systems, embedded systems, mobile computing, flash memory database, and fault-tolerant systems in Anyang University.

Seok-Woo Jang, he received his B.S., M.S., Ph.D. degrees in Computer Science from Soongsil University, Seoul, Korea, in 1995, 1997, and 2000, respectively. He is currently working as an Assistant Professor at Anyang University, Korea. His primary research interests include robot vision, fuzzy systems, augmented reality, video indexing and retrieval, intelligent game, biometrics and pattern recognition.

