Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 712028, 8 pages
http://dx.doi.org/10.1155/2013/712028

Research Article

A Self-Learning Sensor Fault Detection Framework for Industry Monitoring IoT

Yu Liu,[1] Yang Yang,[2] Xiaopeng Lv,[3] and Lifeng Wang[4]

[1] *State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China*
[2] *Information Center of Guangdong Power Grid Corporation, China Southern Power Grid, Guangzhou 510620, China*
[3] *Beijing Guotie Huachen Communication & Infomation Technology Co., Ltd., Beijing 10070, China*
[4] *Beijing Electronic Science and Technology Institute, Beijing 100070, China*

Correspondence should be addressed to Yu Liu; liuyu.paper@gmail.com

Received 8 June 2013; Accepted 3 August 2013

Academic Editor: Zhongmei Zhou

Copyright © 2013 Yu Liu 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.

Many applications based on Internet of Things (IoT) technology have recently founded in industry monitoring area. Thousands of sensors with different types work together in an industry monitoring system. Sensors at different locations can generate streaming data, which can be analyzed in the data center. In this paper, we propose a framework for online sensor fault detection. We motivate our technique in the context of the problem of the data value fault detection and event detection. We use the Statistics Sliding Windows (SSW) to contain the recent sensor data and regress each window by Gaussian distribution. The regression result can be used to detect the data value fault. Devices on a production line may work in different workloads and the associate sensors will have different status. We divide the sensors into several status groups according to different part of production flow chat. In this way, the status of a sensor is associated with others in the same group. We fit the values in the Status Transform Window (STW) to get the slope and generate a group trend vector. By comparing the current trend vector with history ones, we can detect a rational or irrational event. In order to determine parameters for each status group we build a self-learning worker thread in our framework which can edit the corresponding parameter according to the user feedback. Group-based fault detection (GbFD) algorithm is proposed in this paper. We test the framework with a simulation dataset extracted from real data of an oil field. Test result shows that GbFD detects 95% sensor fault successfully.

1. Introduction

Internet of Things (IoT) has been paid more and more attention by the government, academe, and industry all over the world because of its great prospect [1–3]. In the IoT application field, intelligent industry is an important branch. A wired or wireless sensor network is the basic facility of the industry monitoring IoT. These networks comprising of thousands of inexpensive sensors can report their values to the data center. The aim of the monitoring system is to guarantee the process of production.

Fault detection is an important process for industry monitoring IoT, but it is a difficult and complex task because there are many factors that influence data and could cause faults. And faults are application and sensor type dependent [4–6]. Sensors in an industry monitoring IoT have three features: (1) *Big*: thousands of sensors on different devices are working together, (2) *Multitypes*: many physical quantities are needed to determine the production status, (3) *Uncertainty*: different workload is needed according to the production plan and some devices need shut down for examination. So the values of correlative sensors will change between different levels.

From the data-centric view, we focus on the *Outliers*, *Stuck-at* faults and *Spikes* [7]. From the application and system view, we focus on the rational and irrational trend detections. A rational trend means the sensor value transforms from one level to another smoothly and it is caused by a rational reason, such as shut down a device. An irrational trend means value changed when something is wrong with a device. The typical two mistakes of the monitoring system are taking a rational trend for an *Outlier*, or ignoring an irrational trend while values are still in range.

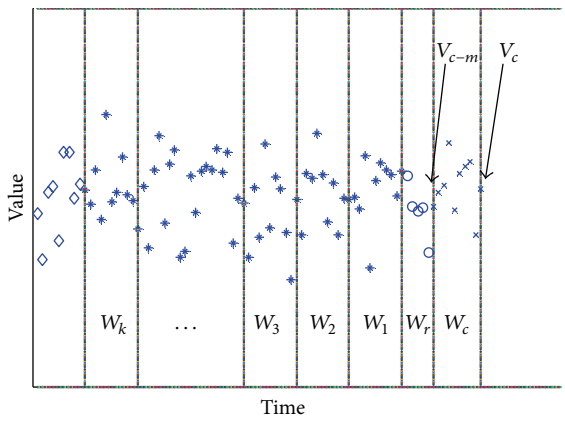


FIGURE 1: Estimation of the data distribution in the Statistics Sliding Windows (SSW) for one time instance.

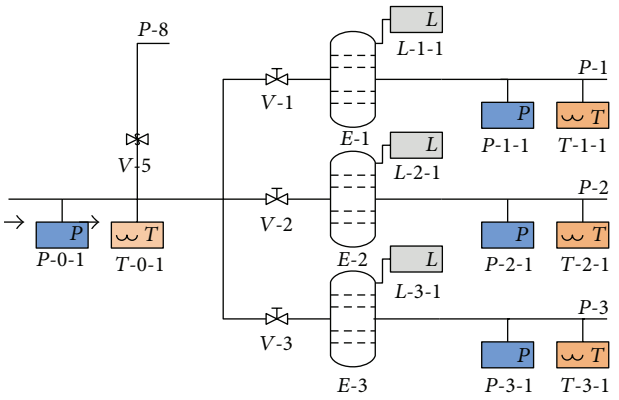


FIGURE 2: Typical flowchat of a gas processing plant.

In this paper, we propose a self-learning sensor fault detection framework for industry monitoring IoT. The data model design is described in Section 3. In Section 4, the framework and the core algorithm are discussed. A simulation experiment base on real data is shown in Section 5.

2. Related Work

Many researchers pay their attention to building a smart monitoring system. Bressan et al. [8] created a solid routing infrastructure through RPL. Castellani et al. [9] concentrate on the actual implementation of the communication technology and presented a lightweight implementation of an EXI library. Yuan et al. [10] present a parallel distributed structural health monitoring technology based on the wireless sensor network. An IoT communication framework for distributed worldwide health care applications is maintained in [11]. All these works are focused on the basic frameworks, protocols, and communication technologies of monitoring systems but discussed less on sensors management.

For modeling sensor network data, Guestrin et al. [12] propose a framework, for the nodes in the network to collaborate in order to fit a global function to each of their local measurements. This is a parametric approximation technique

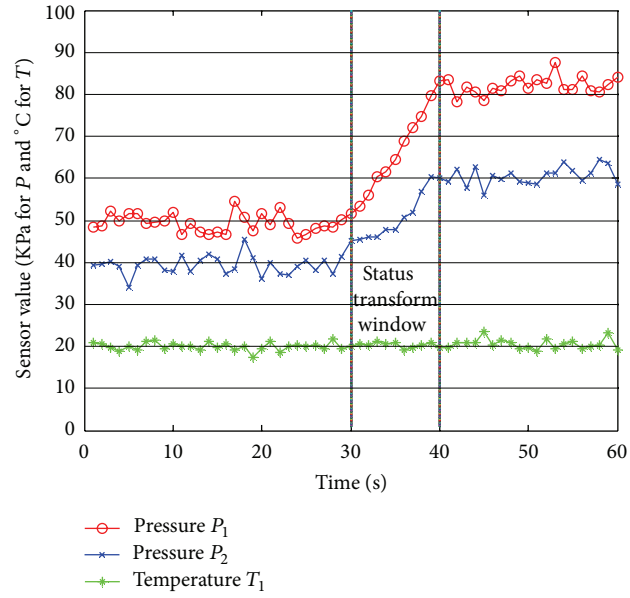


FIGURE 3: Status transformation process for two time instances.

and has more parameters then our approach. References [13, 14] study the problem of computing order statistics in a sensor network. There has also been work on predicting and caching the values generated by the sensors [15, 16], which can result in significant communication savings. But all these approaches are not fit our setting.

A similar approach for sensor fault detection in streaming data is described by Yamanishi et al. [17]. In contrast to our work, their method does not operate on sliding windows but rather on the entire history of the data values. Chan et al. [18] extend the study of algorithms for monitoring distributed data streams from whole data streams to a time-based sliding window, but their focus is on presenting a communication-efficient algorithm.

Ding et al. [19] combined trajectories of all nodes and the paramealgorithm which requires low computational overhead. The proposed algorithm compared its sensor reading with the median value of its neighbors' readings. Gao et al. [20] approach WSN fault detection problems using spatial correlation with the assumption of similar reading within cross range of neighbor nodes. Krishnamachari and Iyengar [21] tried to solve the faulty node detection problem by using localized event region and they assume that the system knows the location of sensor. In an industry monitoring sensor network, finding out a neighbor automatically is very hard. In our approach, we separate sensors into groups according to the production flow charts.

3. Data Model Design

This section focuses on the sensor data model design. We model sensors from the view of value for *Outliers*, *Stuck-at* faults, and *Spikes* detection and from the application view for event detection. The events we are interested in are rational trend and irrational trend.

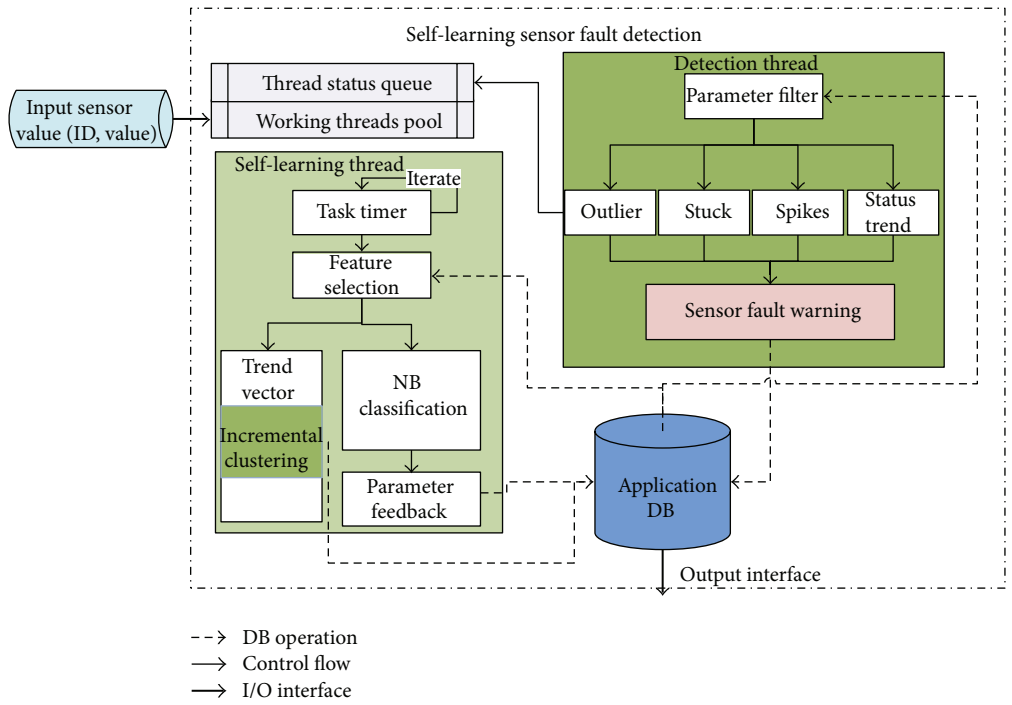


Figure 4: A self-learning sensor fault detection framework.

3.1. Sensor Value. For detection the *Outliers, Stuck-at* faults, and *Spikes*, we propose a statistics method. Figure 1 shows the mechanism of Statistics Sliding Windows (SSW). For sensor s, the current value v_c and the previous m values form the current windows $W_c = \{v_{c-m}, v_{c-m+1}, v_{c-m+2}, \ldots, v_c\}$. In the recent history, we can define k windows with the same length and get the set $H_v = \{W_1, W_2, W_3, \ldots, W_k\}$. We estimate the values in each sliding window by Gaussian distribution, $W_i \rightarrow N(\mu_i, \sigma_i^2)$ (Formula (1)). If $\sigma_c^2 = 0$, a *Stuck-at* fault is detected. And when σ_c^2 is big enough, a *Spikes* fault may happened:

$$W_i \longrightarrow N\left(\mu, \sigma^2\right),$$
$$\mu = \frac{\sum_{k=1}^{m} v_k}{m}, \quad \sigma^2 = \frac{\sum_{k=1}^{m} \left(v_k - \mu\right)^2}{m}. \tag{1}$$

There is a buffer named W_r in front of the current window W_c. With the new samples coming into W_c, the obsolete samples deserted by W_c will become a member of W_r. When the size of W_r reach m, the oldest window in H_v will be discarded and the current W_r will join H_v as W_1. With SSW, large numbers of historical sensor values are regressed to k pairs of Gaussian characteristics. In a real application, we need not hold all the recent values in the memory.

3.2. Status Group. In the industry monitoring IoT, the status of a production line may be uncertain. With the different manufacturing techniques and different workloads, some devices in the production line may be shut down. In this case, the whole production line is still working, but values of sensors monitoring the shutdown devices will run out of range (*Outliers*). At the same time, related devices may also change with the shutting down operation. In an industry monitoring IoT, the rational status transformation of sensors should be recognized and ignored.

Figure 2 shows a typical flow chat of a gas-processing plant. *P-1*, *P-2*, and *P-3* are three parallel subpipelines which are controlled by *V-1*, *V-2*, and *V-3*, respectively. Each subpipeline can be shut down independently and this operation will affect the value of *P-0-1*, a pressure sensor on the main input pipeline. In our approach, we deposit the sensors in the goal-processing plant into several status groups. Although the application background is important to the disposition method, we can follow some common rules as follows:

$$G = \{S_1, S_2, \ldots, S_n\}, \quad S_i \in \text{VALVE}_P, \; n \leq 10. \tag{2}$$

(1) Sensors on the opposite side of a valve are not in the same group. A valve is a typical controller in the industrial IoT. All the devices on the backward position can be shut down by the proper valve. The sensors on the different side may be in different status.

(2) If there are too many sensors which are controlled by one slave, they should be divided into different groups. In our approach, we will not put more than 10 sensors into one status group because the probable status space will expand acutely with the increasing sensor numbers. In this case, sensors can be grouped by their relative position with the most complex device, because the complex device may lead to status change most possibly.

(1) let W^m be the statistics sliding windows size;
(2) let ε_s be the outlier detection threshold for sensor S and let P be the global list to keep all of the ε_s;
(3) let U and V be the global arrays to keep the last 10 gaussian distribution characteristics for each sensor;
(4) let W^n be the status transform windows size and $\theta_i/4$ be the trend vector merging threshold for group G_i, all the rational trend vectors of G_i are stored in R_i;
(5) procedure **GFD ()**
(6) init P, loading ε from Appilcation DB;
(7) init U and V, loading the last 10 distribution characteristics from Application DB;
(8) create C DetectionThread threads, $C = Num_{cpu} * 2$;
(9) start **SelfLearningThread**();
(10) do
(11) get a value set v_i for sensors in a group G_i;
(12) find a idle DetectionThread;
(13) **DetectionThread**(v_i);
(14) while not end;
(15) return;
(16) procedure **DetectionThread**(v_i)
(17) if (**IsStuck**(v_i) or **IsSpikes**(v_i))
(18) return;
(19) if (**IsOutlier**(v_i) and not **IsRatStatChange**(v_i))
(20) return;
(21) **IsRatStatChange**(v_i);
(22) return;
(23) procedure **IsStuck**(v_i)
(24) get ${\sigma_{ij}}^2$ for sensor j by T_c;
(25) if (${\sigma_{ij}}^2 = 0$)
(26) mark s_{ij} as a Stuck;
(27) return;
(28) procedure **IsSpikes**(v_i)
(29) get μ_{ij} and ${\sigma_{ij}}^2$ for sensor j by T_c;
(30) if ($\mu_{ij}/\text{mean}(U(\mu_i)) < \xi$ and ${\sigma_{ij}}^2/\text{mean}(V({\sigma_i}^2)) > \tau$)
(31) mark s_{ij} as a Spikes;
(32) return;
(33) procedure **IsOutlier**(v_i)
(34) use U and V to caculate θ_{ij};
(35) if ($\theta_{ij} > \varepsilon_{ij}$) mark s_{ij} as an Outlier;
(36) return;
(37) procedure **IsRatStatChange**(v_i)
(38) use the last n values of s_i to fit the trend vector S_i;
(39) for each rational trend r_i in R_i
(40) if (cosine(r_i, S_i) < $\theta_i/4$)
(41) mark S_i as a rational status change; $S_{ij} = \text{mean}(S_{ij}, r_{ij})$;
(42) mark the max unstable sensor s_{ij} as a sensor falut;
(43) return;
(44) procudure **SelfLearningThread**()
(45) while (true) do:
(46) load the user feedback;
(47) for each miss alart
(48) if missed Outlier then $\varepsilon_{ij} = \varepsilon_{ij} \div \lambda$;
(49) if missed irrational status change then $\theta_i = \theta_i \times \lambda$;
(50) **IncClustering**(S_i, *null*);
(51) for each false alart
(52) if false Outlier then $\varepsilon_{ij} = \varepsilon_{ij} \times \lambda$;
(53) if false irrational status change then $\theta_i = \theta_i \div \lambda$;
(54) **IncClustering**(S_i, r_{ij});
(55) sleep(T); continues;

Algorithm 1: Continued.

```
(56) return;
(57) procedure IncClustering(S_i, r_ij)
(58)     if r_ij is not null then for each S_ij in S_i;
(59)       if there is cosine(r_i, S_i) < θ_i then S_ij = mean(S_ij, r_ij);
(60)       else add r_ij into S_i;
(61)    for each two S_ij, S_ik in S_i;
(62)         if (cosine(S_ij, S_ik) < θ_i)
(63)              S_ij = mean(S_ij, S_ik); remove S_ik;
(64) return;
```

Algorithm 1: A group-based fault detection Algorithm.

(3) The production line is normally divided into many units according to the geographical position. We can ignore the relationship between sensors in different sections.

For the 11 sensors shown in Figure 2, we divide them into four groups which are $G_0 = \{P_{0-1}, T_{0-1}\}$, $G_1 = \{P_{1-1}, L_{1-1}, T_{1-1}\}$, $G_2 = \{P_{2-1}, L_{2-1}, T_{2-1}\}$, and $G_3 = \{P_{3-1}, L_{3-1}, T_{3-1}\}$. G_0 represents the status of the main input pipeline. G_1, G_2, and G_3 are status groups which are derived from the three parallel subpipelines respectively.

3.3. Status Model. For a status group $G = \{S_1, S_2, \ldots, S_n\}$, all the sensors in G may have different trends when the production status changed. Figure 3 shows a status transformation process. P_1, P_2, and T_1 are stable in the first 30 and the last 20 seconds. The interval from 30 s to 40 s is called the Status Transform Window (STW). P_1 and P_2 increase to a new level while T_1 keep the same trend. And the slopes of P_1 and P_2 are different. We use the trend vector which contains all the slopes of one sensors's group to represent the status transformation, that is, $S = \{k_1, k_2, \ldots, k_n\}$. The trend of a sensor can be found by fitting its values by a specific size of STW. Here, we use the least-square method [22] (Formula (3)) to fit the values and record the rational trend vector by the sensor index. We use the key idea of incremental clustering algorithm [23] to handle the trend vectors, get the cosine angle between the current trend and existing vectors, respectively. If the angle is big enough then mark it a new trend. Otherwise, a repeated trend is found and we only need to merge it with the closest vector. In Section 4, the specific clustering method will be given for details:

$$\widehat{Y} = \alpha + \beta \widehat{X},$$
$$\alpha = \frac{\sum Y}{n} - \frac{\beta \sum X}{n}, \quad \beta = \frac{n \sum XY - \sum X \sum Y}{n \sum X^2 - (\sum X)^2}. \tag{3}$$

4. Design of Architecture

In this section we propose the architecture for sensor fault detection in industry monitoring at first. Then, how to realize our algorithm is discussed.

Table 1: Simulation data description.

Feature	Value
Sampling rate	60 s
Sensor count	5800
Total samples	751.68 million
Status groups	1340
Outliers	437
Stuck-at & *Spikes*	163
RT missed	961
IRT missed	35

4.1. A Self-Learning Framework. The self-learning sensor fault detection architecture is shown in Figure 4. There are four modules in our approach.

(1) Application DB: all the parameters are stored in the Application DB, including the threshold ε_s for sensor s and the history statistics result μ_i, σ_i^2. The grouping information is also serialized in this database. This DB is the interface for high layer application which can get the sensor fault prediction and input the user feedback.

(2) Detection Thread: it is a background service and contains the main detecting process. Since our approach is an online detection, a series of detection thread will be created and maintained by the working thread pool and a related status queue. When new data is coming, the working thread dispatcher a wakes up a pending thread to handle it.

(3) Self-Learning Thread: the self-learning thread uses the OS timer as a driver. The user feedback about the detection result will be rechecked by this thread to revise the trend vectors.

4.2. The GbFD Algorithm. Dividing the sensors into status group is the key idea in our approach. For group-based sensor fault detection, we propose the (group-based fault detection) GbFD algorithm.

The GbFD Algorithm 1 starts by initializing the global parameters (line 6-7); then it instantiates the two core processes *SelfLearningThread* and *DetectionThread*. The input

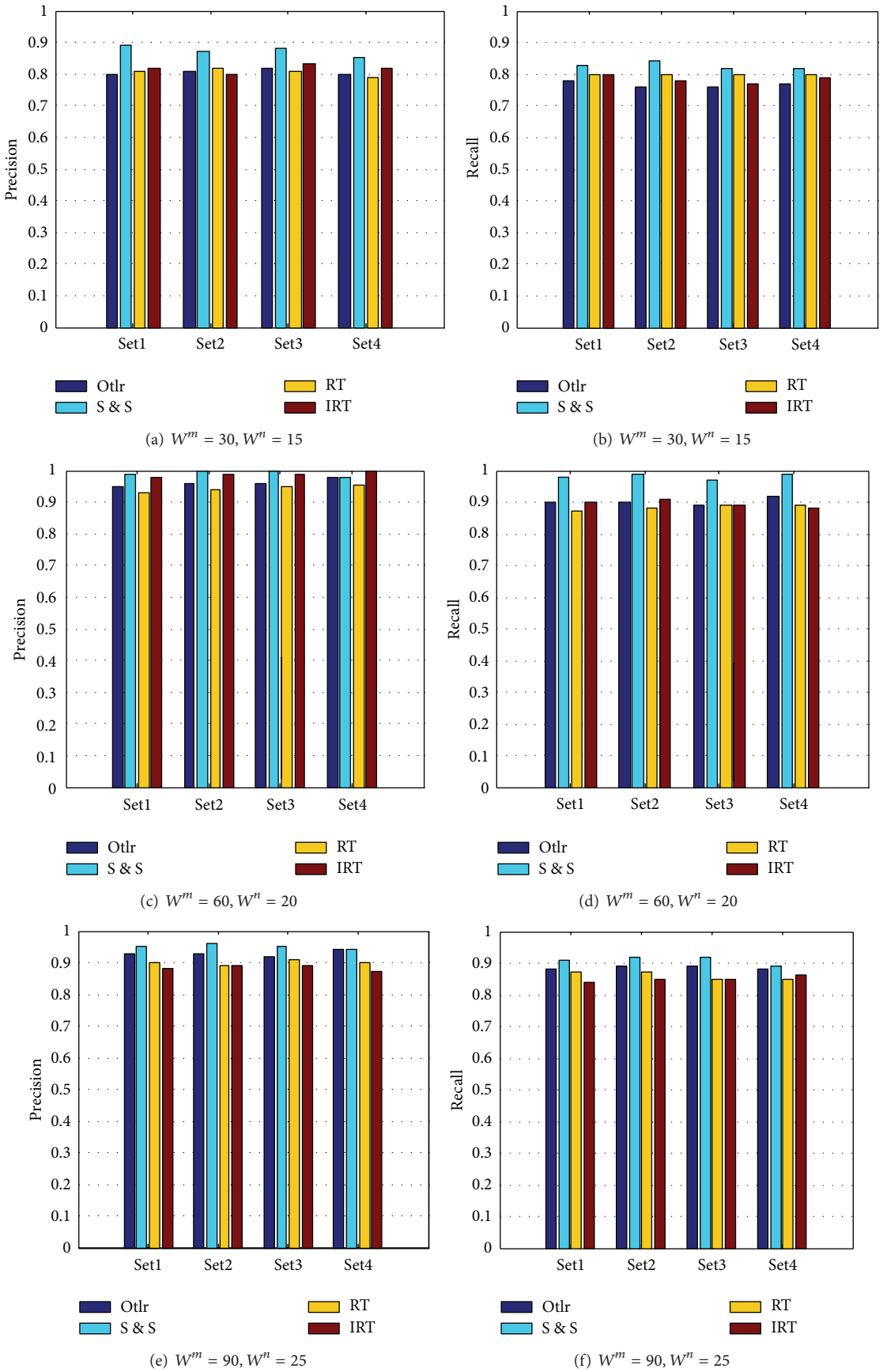


Figure 5: Four-folder cross-validation for precision (a, c, e) and recall (b, d, f) with different W^m and W^n.

of GbFD is defined as a set that contains one time samples for a status group.

In the *DetectionTread*, we first check the *Stuck-at* faults and *Spikes* by calling the *IsStuck* and *IsSpikes* methods. And *IsRatStaChange* is called two times (line 19–21) to detect the sensor status transformation. When an *Outlier* was detected, GbFD should give the conclusion whether it is a rational operation such as shut down the device, and if no *Outlier* happened, we also need to check the abnormal status transformation. The first calling of *IsRatStaChange* is controlled by the *IsOutlier* method with an *AND* logic; no more algorithm complexity is increased by the twice calling. The *SelfLearningThread* works as a background service. Its responsibility is learning the user feedback to find out the missing detection and the false detection and adjusting the relational parameters. For a new rational trend vector, we will add it to the proper group rational trend history by the *IncClustering* method, which can recluster the grouped trend vectors according to the specified angle threshold.

The time complexity of the GbFD algorithm is dominated by *IsOutlier* and *IsRatStuChange* methods. For a sensor in group G, the detection computation takes $O((W^n + d) \times W^m)$, W^n is the STW size, d is the length of G, and W^m is the SSW size. In a real application, the two windows sizes are steerable and d is less than 20 for the most part. Moreover, a proper thread dispatch mechanism will guarantee GbFD to handle the real-time detection task. And, the self-learning process is more complex due to many iterative operations, but it is a background service and does not require real-time performance.

5. Experimental Evaluation

We built an application to evaluate our framework. This application was implemented in JAVA, about 2400 lines code and used Oracle as the Application DB.

5.1. Data Preparation. We use real data from an oil field in China. This oil field has 20 oil/gas treatment plants and all of the production equipments are monitored by sensors which can be mainly classified as temperature sensor, pressure sensor, and liquid level sensor. The sampling rate of the production IoT is 60 seconds. We obtained the sample data of 10,000 sensors between January 1st 8 PM to October 31th 8 AM, 2012. Since the production lines are relatively stable, we filtered the original data by two steps. Firstly, some production units that never make a mistake were eliminated. Secondly, for a plant, we discard some data in its stable period.

Table 1 shows features of our simulation dataset. We choose more than 751 million samples from 5800 sensors. According to the corresponding flow charts, we separate the sensors into 1340 groups. Each group has 4.33 sensors in average, and the maximum group contains 8 sensors. We analyze the user feedback history and find out four typical errors. *Outlier* means that a value runs out of range and it is caused by the sensor failure. We put *Stuck-at* faults and *Spikes* together. The *Rational Trend* (RT) missed fault means that sensors submit an exceptional value after a rational user operation, such as shuting down a device for examination. And the *Irrational Trend* (IRT) missed fault means that something wrong happened with the production line and the sensor values changed with it but still suitable with the threshold condition.

5.2. Experiment. We split the simulation data into four datasets according to the time sequence. Each time we use optional three data sets to train the GbFD algorithm and the remainder is used as the testing data.

We use three pairs SSW size and STW size, which are (30, 15), (60, 20), and (90, 25), to run the four-folder cross-validation test. The result is shown in Figure 5. When $W^m = 30$ and $W^n = 15$, each cross get a precision of about 80%. This result is not good enough. A satisfactory result is generated in the next test; $W^m = 60$ and $W^n = 20$ get a mean 95% accuracy. But with the increase of two windows size, the accuracy of GbFD will go to an opposite direction. This phenomenon indicates that the sizes of SSW and STW are strong correlating with our algorithm. Choosing the proper windows size can increase the detection sensitivity and the windows size (60, 20) is suitable with our simulation data.

6. Conclusions

We present a self-learning sensor fault detection framework in this paper. We propose a model which can represent the sensor value, sensor relationship, and sensor status transformation. GbFD algorithm is proposed to detect the sensor fault. And we use real data from an oil field for validation. Experimental results show that our system can detect 95% of data fault in the simulation data which contains 751.68 million samples from 5800 sensors.

We will continue validate our approach on other dataset to find out the proper statistical sliding window size and status transform windows size in different application contexts. Our goal is to build a sensor health management system for industry IoT that includes not only sensor fault detection, but also sensor lifecycle prediction and sensor inspection management.

Acknowledgments

The authors would like to thank Jun Ma for helping them to dispose the simulation data. This research is supported by the National Natural Science Funds of China (61202238), the State Key Laboratory of Software Development Environment Funds (SKLSDE-2011ZX-08), and the Fundamental Research Funds for the Central Universities (2012RC0209).

References

[1] Y. Liu and G. Zhou, "Key technologies and applications of internet of things," in *Proceedings of the 5th International Conference on Intelligent Computation Technology and Automation (ICICTA '12)*, pp. 197–200, Zhangjiajie, China, January 2012.

[2] W. Baoyun, "Review on Internet of Things," *Journal of Electronic Measurement and Instrument*, vol. 23, no. 12, pp. 1–7, 2009.

[3] S. Haller, S. Karnouskos, and C. Schroth, "The internet of things in an enterprise context," in *Future Internet*, vol. 5468 of *Lecture Notes in Computer Science*, pp. 14–28, Springer, Berlin, Germany, 2009.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–105, 2002.

[5] L. Paradis and Q. Han, "A survey of fault management in wireless sensor networks," *Journal of Network and Systems Management*, vol. 15, no. 2, pp. 171–190, 2007.

[6] Y. Zhang, N. Meratnia, and P. Havinga, "Outlier detection techniques for wireless sensor networks: a survey," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 2, pp. 159–170, 2010.

[7] K. Ni, N. Ramanathan, M. N. H. Chehade et al., "Sensor network data fault types," *ACM Transactions on Sensor Networks*, vol. 5, no. 3, pp. 1–29, 2009.

[8] N. Bressan, L. Bazzaco, N. Bui, P. Casari, L. Vangelista, and M. Zorzi, "The deployment of a smart monitoring system using wireless sensor and actuator networks," *Proceedings of International Conference on Smart Grid Communications*, pp. 49–54, 2010.

[9] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web services for the Internet of things through CoAP and EXI," in *Proceedings of the IEEE International Conference on Communications Workshops (ICC '11)*, pp. 1–6, Kyoto, Japan, June 2011.

[10] S. Yuan, X. Lai, X. Zhao, X. Xu, and L. Zhang, "Distributed structural health monitoring system based on smart wireless sensor and multi-agent technology," *Smart Materials and Structures*, vol. 15, no. 1, pp. 1–8, 2006.

[11] N. Bui and M. Zorzi, "Health care applications: a solution based on the Internet of Things," in *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL '11)*, article 131, October 2011.

[12] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN '04)*, pp. 1–10, Berkeley, Calif, USA, April 2004.

[13] J. Considine, M. Hadjieleftheriou, F. Li, J. Byers, and G. Kollios, "Robust approximate aggregation in sensor data management systems," *ACM Transactions on Database Systems*, vol. 34, no. 1, article 6, 2009.

[14] M. B. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '04)*, pp. 275–285, Paris, France, June 2004.

[15] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive stream resource management using Kalman Filters," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*, pp. 11–22, Paris, France, June 2004.

[16] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 563–574, June 2003.

[17] K. Yamanishi, J.-I. Takeuchi, and G. Williams, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*, pp. 320–324, August 2000.

[18] H.-L. Chan, T.-W. Lam, L.-K. Lee, and H.-F. Ting, "Continuous monitoring of distributed data streams over a time-based sliding window," *Algorithmica*, vol. 62, no. 3-4, pp. 1088–1111, 2012.

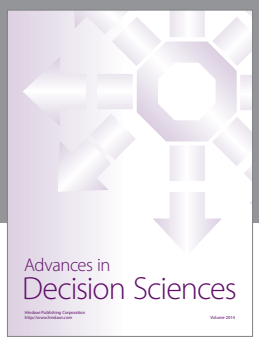
[19] M. Ding, D. Chen, K. Xing, and X. Cheng, "Localized fault-tolerant event boundary detection in sensor networks," in *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '05)*, pp. 902–913, March 2005.

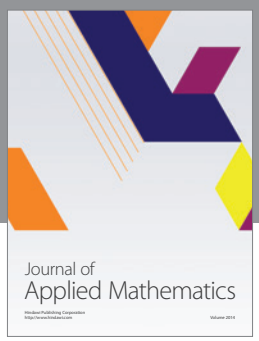
[20] J.-L. Gao, Y.-J. Xu, and X.-W. Li, "Weighted-median based distributed fault detection for wireless sensor networks," *Journal of Software*, vol. 18, no. 5, pp. 1208–1217, 2007.

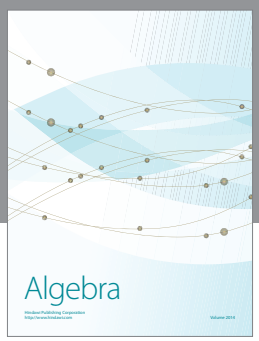
[21] B. Krishnamachari and S. Iyengar, "Distributed Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 241–250, 2004.

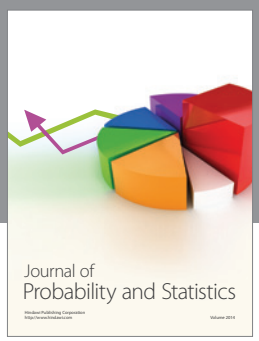
[22] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society For Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

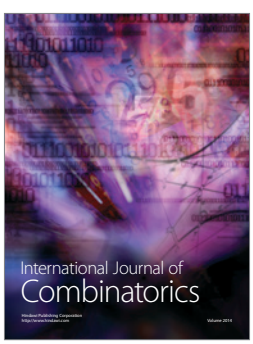
[23] X. Su, Y. Lan, R. Wan, and Y. Qin Y, "A fast incremental clustering algorithm," in *Proceedings of the International Symposium on Information Processing (ISIP '09)*, pp. 175–178, Huangshan, China, 2009.

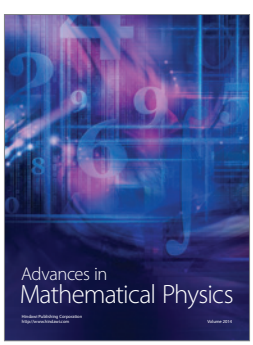


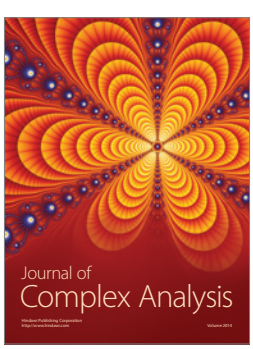


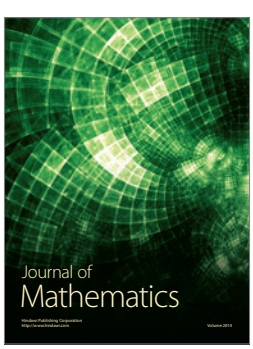


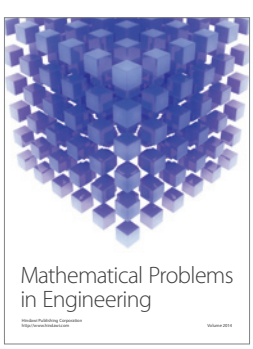


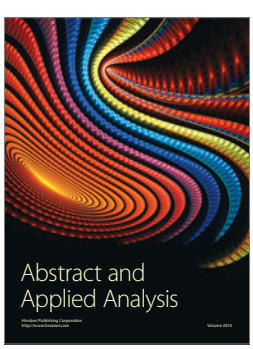


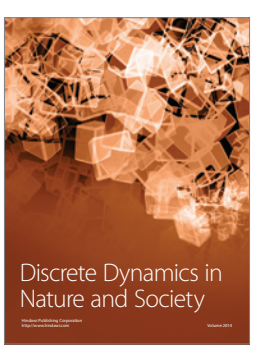


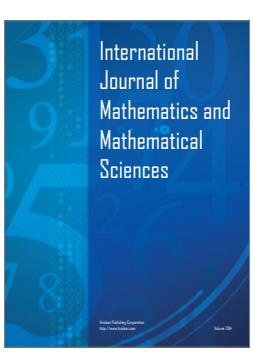












Hindawi

Submit your manuscripts at
http://www.hindawi.com