A Weighted Moving Average-based Approach for Cleaning Sensor Data

Yongzhen Zhuang[†], Lei Chen[†], X. Sean Wang[‡], Jie Lian[*]
[†] Department of CSE, Hong Kong University of Science and Technology, Hong Kong
[‡] Department of CS, University of Vermont, Burlington, Vermont
[*] Department of E&CE, University of Waterloo, Waterloo

Abstract

Nowadays, wireless sensor networks have been widely used in many monitoring applications. Due to the low quality of sensors and random effects of the environments, however, it is well known that the collected sensor data are noisy. Therefore, it is very critical to clean the sensor data before using them to answer queries or conduct data analysis. Popular data cleaning approaches, such as the moving average, cannot meet the requirements of both energy efficiency and quick response time in many sensor related applications.

In this paper, we propose a hybrid sensor data cleaning approach with confidence. Specifically, we propose a smart weighted moving average (WMA) algorithm that collects confidence data from sensors and computes the weighted moving average. The rationale behind the WMA algorithm is to draw more samples for a particular value that is of great importance to the moving average, and provide higher confidence weight for this value, such that this important value can be quickly reflected in the moving average. Based on our extensive simulation results, we demonstrate that, compared to the simple moving average (SMA), our WMA approach can effectively clean data and offer quick response time.

1 Introduction

In recent years, sensor networks as an emerging distributed system have been widely used in monitoring various environmental parameters [1, 18]. However, due to different reasons, such as the low quality of sensing devices and random effects of external sources [4], sensor data are considered to be noisy. How to remove this noise or at least reduce the effect that brought about by the noise is a key issue to answer queries [9] or detect events [17] accurately.

One popular approach to remove noise in random samples and compute the monitoring values is to use a moving average [7, 5]. Unlike a moving average that is usually used for a one-dimensional time series, moving average in sensor networks has two dimensions. Sensor data are averaged temporally within one sensor, and also spatially among neighboring sensors. A *simple moving average* (SMA) algorithm for sensor networks is as follows. At any time t, the algorithm first averages a sequence of samples $x_{i,t-k+1}, \ldots, x_{i,t}$ at each sensor i, and gets $\bar{x}_i = (x_{i,t-k+1} + \ldots + x_{i,t})/k$. It then averages values of neighboring sensors, $\overline{\bar{x}}_i = \Sigma_{j \in R(i)} \overline{x_j} / |R(i)|$, where $R(i)$ is a set of neighboring sensors of sensor i.

However, SMA is not suitable for sensor network applications, when taking into account the two important evaluation criteria: energy efficiency and query response time. In the SMA method, both criteria cannot be met at the same time. In order to improve energy efficiency, sampling rates should be lowered (i.e. the interval between two consecutive samples are lengthened). The consequence of low sampling rates is that it takes a long time to reflect a change in the moving average. On the other hand, if the sampling rates are high, the response to a change can be quick. However, more samples need to be taken and we know that sampling is one of the costly operations in sensors [10].

In this paper, we propose a *weighted moving average* (WMA) algorithm, that collects confidence data from sensors and computes the weighted moving average. In particular, the *temporal moving average* is defined as:

$$\bar{x}_i^t = \frac{w_{i,t-k+1} x_{i,t-k+1} + \ldots + w_{i,t} x_{i,t}}{w_{i,t-k+1} + \ldots + w_{i,t}},$$

where $w_{i,t}$ is the weight of value $x_{i,t}$ at sensor i and timestamp t, which is related to the confidence of this value. We also consider the spatial moving average of those spatially correlated sensors. The *spatial moving average* is

$$\bar{x}_i^s = \frac{\Sigma_{j \in Neighbor(i)} b_{j,t} w_{j,t} x_{j,t}}{\Sigma_{j \in Neighbor(i)} b_{j,t} w_{j,t}},$$

where weight $b_{j,t}$ is a boolean value to decide whether or not a value of a neighboring node needs to be included in the moving average. The rationale behind the WMA algorithm is to let sensors report confidence value $x_{i,t}$ to the sink.

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

We adopt low sampling rates when $x_{i,t}$ is smooth and predictable. Only when $x_{i,t}$ jumps out of the prediction range, do we increase the sample size and obtain high confidence $x_{i,t}$. We also give more weight for this out-of-range value, so that it can be quickly reflected in the weighted moving average. Previous works clean data either in the sink [7] or within a sensor [4]. In our approach, data cleaning is performed on both sides. On the sensor side, we use multiple sampling to remove random noise from data, whereas on the sink side, we use the weighted moving average in both temporal and spatial dimensions to further smooth the data.

The contributions of this paper are threefold:

- We propose a *weighted moving average* (WMA) algorithm, which can provide clean data for applications in sensor networks with high energy efficiency and a short response time.
- We present a prediction and testing approach to evaluate values reported from sensors to the sink, and increase the weight for a value when it is of great importance to the moving average.
- We provide an effective method that utilizes the spatial information from neighbors to efficiently reduce the sampling rates of sensors. In contrast, without using such neighboring information, a sensor has to draw many samples at a high rate in order to increase the confidence of its out-of-range random sample.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce the *weighted moving average* (WMA) algorithm. In Section 3, we discuss the three steps of the WMA. In Section 4, we present the experimental results. Section 5 describes the related works, followed by the conclusion in Section 6.

2 Overview

In noisy environments, the values detected by sensors are random samples of true environmental parameters. A query might say "find sensors in an area whose temperature are between 30C - 40C", "return sound sensors whose volume are higher than 30dB", and so on. As long as the moving average of the sensor time series has been computed, these queries can be answered using the clean data provided by the moving average.

We introduce the concept of confidence in calculating the moving average. Each value $x_{i,t}$[1] reported from individual sensors can be associated with a confidence, which is related to the number of samples - more samples gain higher confidence. More formally, if the noise variance of sensor i is σ_i^2 and k samples are drawn, the variance of the average $\bar{x}$ is σ_i^2/k. Given an error range $\pm e$, the confidence of $\bar{x}$ is the cumulative probability in the interval

[1] A value in the moving average can be a single sample, or the combination of multiple samples.

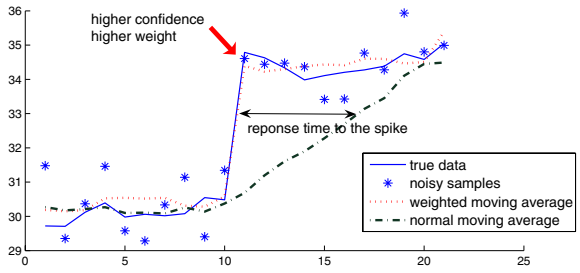


Figure 1. comparing weighted moving average with normal moving average

$[-\sqrt{k}e/\sigma_i, +\sqrt{k}e/\sigma_i]$ of the *normal distribution* $N(0,1)$, which is increasing with respect to k. If a value has a higher confidence, we assign it more weight in computing the moving average.

We notice that it is too costly to let all sensors sample abundantly at any time to improve their confidence. The idea is to get more samples and increase the confidence of a particular value only if it is *important* and might affect the moving average. When the *important* values are assigned more weight, the weighted moving average is closer to the true data than the common moving average. The important values can be identified by looking at the sensor time series. If the trend of a sensor is smooth and slow, we can use a single sample and the same weight for each value to compute the moving average accurately. However, if there exists a spike, we have to collect more samples at that point. Since more samples provide higher spike confidence, we can therefore increase the weight of the value at the spike when computing the moving average. By this means, a spike can be quickly and accurately reflected in the weighted moving average. Data cleaning is performed on two sides, that is, both in the sensor and the sink. In each individual sensor, we adapt the sample rate to obtain more samples when the change is significant and probably cannot be captured quickly without increasing its weight. It can therefore remove most of noise on the sensor side, and send back the values with high confidence to the base station. On the other hand, if the change is small and predictable, we only send the raw sample (the return value is a single sample) back to the sink to save the sampling cost. Although a single sample is noisy, noise contained in these random samples is removed by the moving average. Figure 1 compares SMA with WMA. In particular, the SMA method has a delay in capturing the spike. In contrast, the WMA algorithm can quickly reflect the spike in the weighted moving average. The weighted moving average is energy-efficient since most of the time sensors sample at low rates - one sample per value. They only request more samples from a particular value that is likely to affect the moving average.

To further reduce the number of samples, we make use of the neighboring relationship. In the smooth environment,

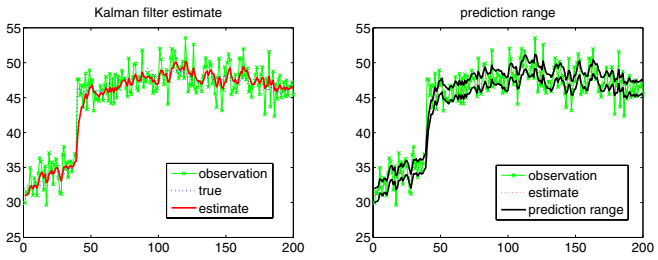


Figure 2. Kalman filter

the change in neighboring sensors is also smooth, and they either have similar patterns or are correlated. In many situations, an *important* value appearing in a sensor may also appear in the other nearby sensors. For example, a fire may affect an area including more than one sensor. All the sensors in the fire area can detect the sudden increase in temperature. Therefore, whenever a sensor detects a suspicious *important* value, it double-checks it with its neighbors. If this value is also detected by neighbors, the confidence is increased. For instance, a sensor detects a fire with the temperature $> 80°$C and a confidence of 60%, and it also finds that its neighbors detect the same high temperature with a confidence of 60%. Then, the sensor can infer the fire with a high confidence. The more neighbors that detect the fire, the higher the confidence. One advantage of using neighboring information is that, a sensor does not need to obtain that many samples to achieve a high confidence level. Instead, it can gain confidence from its neighboring information. Furthermore, if a sensor has low noise variance and can offer a high confidence value yet at a low sampling cost, all its neighbors can rely on this sensor to increase their confidence rather than sampling abundantly by themselves.

3 Weighted Moving Average (WMA)

The WMA algorithm consists of three steps:

- locate *important* values by a range prediction;
- gain confidence for *important* values through sensor testing and neighbor testing at individual sensors;
- perform *weighted moving average* (WMA) at the sink.

Since the communication energy cost per byte is much larger than the computation cost per instruction by a factor of about 1000 [8], the energy cost in range prediction (either using kalman filter or regression) and neighboring testing is negligible. In this section, we introduce the above three steps one by one.

3.1 Range Prediction

A range prediction computes a range for the next timestamp from the current and previous samples. A prediction range for sensor i at timestamp t is denoted as $[l_{i,t}, u_{i,t}]$. If the new sample $v_{i,t}$ falls within the range, the raw sample is sent to the sink (i.e. $x_{i,t} = v_{i,t}$ in the moving average). When the new sample is outside of range, however, we do not know if it is caused by random noise or an environmental change that cannot be captured by predictions. In this case, we have to obtain more samples and conduct tests. We identify a change whenever we have enough confidence to prove that the new value indeed jumps outside of the prediction range. We then send the sample average, as well as its confidence level to the sink. If this value is finally proved to be in the prediction range after taking more samples, we only send the proved value (without attaching the confidence) to the sink. Here, we use two methods for the prediction: Kalman filter and linear regression. Both approaches estimate a value m_t for the next timestamp t. The prediction range is an $\pm e$ error range around m_t, that is,

$$[m_t - e, m_t + e],$$

where $\pm e$ is a given error range.

3.1.1 Kalman Filter

A Kalman filter is a probability-based digital filter which is capable of filtering "white noise". The basic 1st order implementation of Kalman filter can be used to smooth a noisy sensor input for further processing. Figure 2-A demonstrates the smoothing effect. The observation is the noisy samples with high variance, the smoother dotted line is the true data, and the bold line is the estimated value from the Kalman filter. The $\pm e$ estimation range is shown in Figure 2-B.

A Kalman filter is represented by a system model:

$$x_t = A_{t-1}x_{t-1} + q_{t-1},$$

which calculates state x_t from its previous state, and a measurement model is defined as:

$$z_t = H_t x_t + r_t,$$

which describes the relationship of observation z_t and the estimated state x_t. A and H are model parameters, which are identity matrices. The system error and measurement error are $q_{t-1} \sim N(0, Q_{t-1})$ and $r_t \sim N(0, R_k)$, respectively. Given a prior distribution $x_0 \sim N(m_0, P_0)$, Kalman filter estimates m_t and the variance P_t through two recursive phases: *predict* and *update*. The *predict* phase produces an estimate of the current state from the previous timestamp.

$$p(x_t|z_{1:t-1}) = N(x_t|m_t^-, P_t^-)$$

In the *update* phase, a new measurement of the current timestamp is used to refine this prediction.

$$p(x_t|z_{1:t}) = N(x_t|m_t, P_t)$$

More about kalman filter please refer to [19].

3.1.2 Linear Regression

Linear regression is an approach to determine the relationship between two random variables. In our problem, one random variable is timestamp t, and the other is the estimated value m_t. The linear regression model postulates that $m_t = at + b$. The coefficients a and b are determined by the condition that the squared sum of the vertical distances from points to line is minimized.

3.2 Local Testing

3.2.1 Sensor Testing

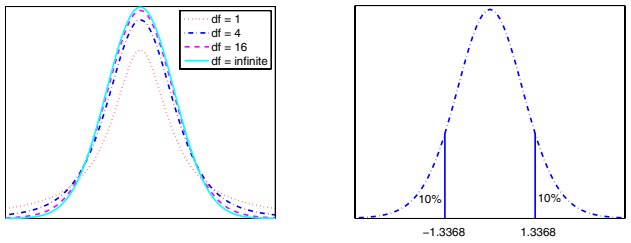
If a sample is not in the prediction range, we need to distinguish between a random error and a real environmental change. We take multiple samples. If the out-of-range sample is due to a random error, we would finally obtain a sample mean in the prediction range. If it is a real environmental change, we gain more confidence about the change by these samples, and can return a more accurate value with a higher confidence.

We gain confidence through hypothesis tests. If a sample $v_{i,t}$ is not in the prediction range $[l_{i,t}, u_{i,t}]$, we get more samples and test if the sample mean $\overline{v}_{i,t}$ is in the range. Usually, the intended answer is modeled as an alternative hypothesis. Therefore, the null hypothesis in this test is H_0 : $\mu_{i,t} < l_{i,t}$ *or* $\mu_{i,t} > u_{i,t}$, where $\mu_{i,t}$ is the true value. This hypothesis can be further divided into two one-side hypothesis. The tests are performed by computing their test statistics

$$T_l = \frac{\overline{v}_{i,t} - l_{i,t}}{\sqrt{\hat{\sigma}_{i,t}}} \text{ and } T_u = \frac{\overline{v}_{i,t} - u_{i,t}}{\sqrt{\hat{\sigma}_{i,t}}}.$$

In t-test, the test statistics follow t distribution of different degrees of freedom. A rejection range with a certain confidence c can be calculated from t distribution. If the size of the test is N (i.e. N samples are drawn), we select the t distribution with the degree of freedom $N-1$. The rejection range of T_l is $T_l > z_l$, where z_l is the one-tail cut-off point in the t distribution and the cumulative probability above z_l is $1-c$. Similarly, the rejection range of T_u is $T_u < z_u$, where z_u is the cut-off point and the cumulative probability below z_u is $1-c$. Figure 3-A shows a family of t distribution with different degrees of freedom (df). Figure 3-B shows the two cut-off points z_l and z_u in a t distribution (df=16) with confidence $c = 90\%$.

The testing algorithm is as follows. The above t-test is repeated once a new sample is drawn. If the null hypothesis H_0 is rejected, the test stops at a value in the prediction range. Therefore, we conclude that the out-of-range sample is caused by the random error, and send the sample mean to the sink. Otherwise, we keep testing until the sample mean is in an $\pm e$ error range with a given confidence. The sample



A. t distribution of different df B. t distribution with cutoff points

Figure 3. t distribution

Test when the first sample $v_{i,t}$ is not in the prediction range $[l_{i,t}, u_{i,t}]$
Step 1: Get one more sample;
Step 2: Compute sample mean $\overline{v}_{i,t}$ and sample variance $(\hat{\sigma}^{\star}_{i,t})^2$;
Step 3: Perform hypothesis test with $H_0 : \mu_{i,t} < l_{i,t}$ *or* $\mu_{i,t} > u_{i,t}$
Step 3.1: If H_0 is rejected, return $\overline{v}_{i,t}$ and go to END;
Step 4: Check the confidence of the error range $\pm e$.
Step 4.1: If $z\hat{\sigma}^{\star}_{i,t} < e$, return $\overline{v}_{i,t}$ and confidence c, and go to END;
Step 4.2: Else, go to Step 1.
END

Table 1. Local test of the important value that falls outside of the prediction range

mean $\overline{v}_{i,t}$ and sample variance $\hat{\sigma}^2_{i,t}$ for sensor i at time t are computed from a set of N samples. Multiple samples can improve the confidence, since the variance of the sample mean is decreasing with the sample size N:

$$(\hat{\sigma}^{\star}_{i,t})^2 = \frac{\hat{\sigma}^2_{i,t}}{N}.$$

We accept the sample mean $\overline{v}_{i,t}$ as a return value whenever the true value falls into the $\pm e$ error range with at least confidence c. In other words, the cumulative probability of Gaussian distribution $N(\overline{v}_{i,t}, (\hat{\sigma}^{\star}_{i,t})^2)$ in range $[\overline{v}_{i,t} - e, \overline{v}_{i,t} + e]$ is higher than c. This can be tested by finding the confidence c cut-off point in the *normal distribution* $N(0,1)$, denoted as z, and accept the sample mean if $z\hat{\sigma}^{\star}_{i,t} < e$. Details of the local testing are described in Table 1.

3.2.2 Neighbor Testing

Neighbor testing is performed during the mean time of the local testing. If a sample falls outside of the prediction range, we check if its neighbors also detect similar abnormal samples. We can conclude with a higher confidence that the abnormal sample is not due to a random error if more neighbors have similar samples.

To use the neighboring information in the testing, a sensor shares its sample mean, sample variance, and sample size with the neighboring sensors. The sample mean and variance of sensor i are $\overline{v}_{i,t}$ and $\hat{\sigma}^2_{i,t}$, respectively, and the

sample size is denoted as $N_{i,t}$. We use the neighboring information of sensor i to improve and speed up the testing. If all neighbors of sensor i share their sample mean, variance, and size, the neighboring information collected by sensor i is

$$\{(\overline{v}_{j,t}, \hat{\sigma}^2_{j,t}, N_{j,t})\}, \quad j \in Neighbor(i).$$

A neighbor is included in the testing if its sample mean $\overline{v}_{j,t}$ is in the $\pm e$ error range of sensor i, which implies that it is very possible that sensor i and j are affected by the same event.

$$\overline{v}_{j,t} \in [\overline{v}_{i,t} - e, \overline{v}_{j,t} + e]$$

We let $Neighbor'(i)$ denote a set of neighboring sensors included in the test.

The idea of neighboring test is to use the neighboring samples to increase the confidence of the current sensor. The new confidence interval from the adjusted variance $z\dot{\sigma}_{i,t}$ is

$$[\overline{v}_{i,t} - z\dot{\sigma}_{i,t}, \overline{v}_{i,t} + z\dot{\sigma}_{i,t}]$$

where $\dot{\sigma}_{i,t}$ is

$$\dot{\sigma}_{i,t} = \min(\frac{\hat{\sigma}^2_{i,t}}{N_{i,t}}, \frac{N_{i,t}\hat{\sigma}^2_{i,t} + \Sigma_{j \in Neighbor'(i)} N_{j,t}\hat{\sigma}^2_{j,t}}{(N_{i,t} + \Sigma_{j \in Neighbor'(i)} N_{j,t})^2})$$

The above formula uses samples from the neighboring sensors, as well as samples from sensor i to reduce the variance of sensor i. Eventually, if $z\dot{\sigma}_{i,t} \leq e$, we say that we have found a sample mean whose error range is less than $\pm e$ with confidence higher than c. If a sensor passes the neighboring test, its sample mean $\overline{v}_{i,t}$ is sent to the sink, as well as a confidence $c_{i,t}$ which is the probability that data has fall within the interval $[\overline{v}_{i,t} - e, \overline{v}_{i,t} + e]$ in distribution $N(\overline{v}_{i,t}, \hat{\sigma}_{i,t}/N_{i,t})$. The consequence is, without neighboring test, each out-of-range change needs to be tested independently by individual sensors until it reaches confidence c. With neighboring information, we can probably stop at a lower confidence $c_{i,t} < c$, and do not need that many samples in the test. In the next section, we describe how to use the neighboring information in the calculation of the *weighted moving average* (WMA).

To save transmission costs, a sensor only shares its information with neighbors when it reaches confidence c. No matter what, the sensor needs to send the sample mean to the sink at this time. It does not cost more messages to share this information with neighbors since they share the same communication channel. Sample variance and samples size are attached to this message. When the parent node forwards the message to an upper level, it can delete attached sample variance and sample size from the message. Thus, the transmission cost of the neighboring test is very low.

3.3 Weighted Moving Average

Weighted moving average (WMA) is computed when the sink obtains the values sent from sensors of two dimensions. The temporal dimension averages values at different timestamp but from the same sensor to remove noise. The spatial dimension averages high confidence values from neighbors at the same timestamp to improve the confidence. There are three kinds of values from sensors:

- a single sample mean $x_{i,t} = \overline{v}_{i,t}$, sometimes it is only one raw sample;
- a sample mean $x_{i,t} = \overline{v}_{i,t}$ with confidence c;
- a sample mean $x_{i,t} = \overline{v}_{i,t}$ with confidence $c_{i,t}$.

We assign a weight $w_{i,t}$ to each value $x_{i,t}$ according to its confidence. If $x_{i,t}$ is not attached by any confidence, we let $w_{i,t} = 1$. If $x_{i,t}$ is attached by confidence c, we magnify its influence in the moving average. We let $w_{i,t} = w_{max}$ for this value, where w_{max} is a predefined maximum weight. If $x_{i,t}$ is attached by confidence $c_{i,t} < c$, we scalarly assign a weight to it: $w_{i,t} = (w_{max} - 1) \cdot c_{i,t}/c + 1$.

The computation of WMA at the sink includes two parts: a temporal average and a spatial average. The temporal part is a weighted version of the normal moving average

$$\bar{x}^t_i = \frac{\hat{w}_{i,t-k}x_{i,t-k} + \ldots + \hat{w}_{i,t}x_{i,t}}{h},$$

where k is a given window size and h is the accumulated temporal weight $h = \hat{w}_{i,t-k} + \ldots + \hat{w}_{i,t}$. To remove the influence of a highly weighted value in future timestamps, we use the most recent weight strategy. Only the right-hand most > 1 weight in $w_{i,t-k}, \ldots, w_{i,t}$ is used in $\hat{w}_{i,t-k}, \ldots, \hat{w}_{i,t}$. Other $\hat{w}_{i,t}$ are set to be *1*.

The spatial part includes value $x_{j,t}$ from sensors j, where $x_{j,t}$ is in the error range of $x_{i,t}$, $[x_{i,t} - e, x_{i,t} + e]$, and it has a high confidence with its weight $w_{j,t} > 1$. That means $x_{j,t}$ is in $x_{i,t}$'s error range and has a high confidence to improve the moving average at $x_{i,t}$. We need to include the spatial values because in the neighboring test, some of the sensors stop at a lower confidence and they need to use their neighbors' values and their higher confidence to improve their moving average. The spatial part of the moving average is

$$\bar{x}^s_i = \frac{\Sigma_{j \in Neighbor(i)} b_{j,t}w_{j,t}x_{j,t}}{m},$$

where m is the accumulated spatial weights:

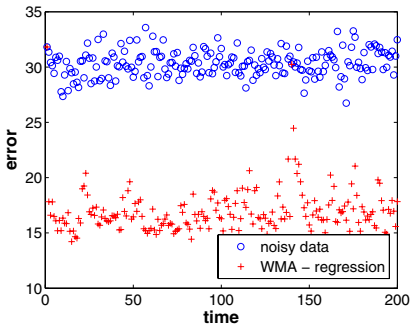
$$m = \Sigma_{j \in Neighbor(i)} b_{j,t}w_{j,t}$$

and $b_{j,t}$ is a boolean variable decide if a neighbor is used in the spatial average or not:

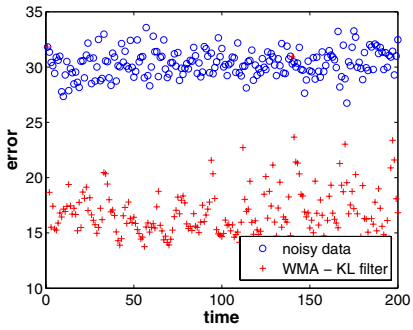
$$b_{j,t} = \begin{cases} 1 & \text{if } x_{j,t} \in [x_{i,t} - e, x_{i,t} + e] \text{ and } w_{j,t} > 1 \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the *weighted moving average* is the combination of the temporal and spatial parts:

$$\bar{x}_{i,t} = \frac{h\bar{x}^t_i + m\bar{x}^s_i}{h + m}.$$



A. Error of regression



B. Error of KL filter

Figure 4. Clean effectiveness

4 Simulation Results

To evaluate the *weighted moving average* (WMA) algorithm, we build our own sensor network simulator. Sensors are placed in a 30×30 square sensor grid with one sensor in each square. With one sensor in each square, neighbors of each sensor include its left, right, top, bottom, top-left, top-right, bottom-left, and bottom right-hand-side sensors. In more complex cases, if the correlation information of sensors is known as a prior from the physical placement, we can make use of it to identify neighbors in the neighboring test. That means each sensor knows exactly who are the neighbors and whose information may be usable. Neighboring sensors share the same communication channel, and can therefore overhear messages sent by other neighbors. Although data aggregation for queries is important, it is not the focus of this paper. Therefore, we simply consider a data collection application. However, note that our WMA approach can be easily modified to be used in other query aggregation applications. For example, in a SUM query, the weighted moving average can be computed within individual sensors, and provide clean data for calculating the aggregated SUM.

The simulation data are from a real time-series data set. We first assign the time series to some randomly placed *initiator sensor nodes*. The time series of the other sensors is a linear combination of the initiator sensors around it, where the closer initiator sensors have higher weights. In this way, we set up the spatial similarity in the network. As we did in [3], we generate noisy data by filling the data into a re-sampling process, which adds different amounts of Gaussian white noise to different sensors. The simulation lasts for 200 timestamps. Sensors are loosely synchronized and data collection is performed at each timestamp. Various strategies, such as aggregation and compression, can be applied to reduce the communication cost during the data collection, which, however, is not of interest in this paper.

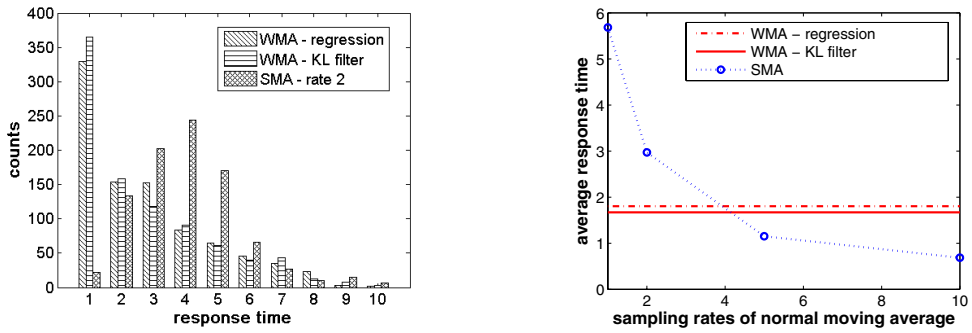
4.1 Clean Effectiveness

We first evaluate the effectiveness of data cleaning by the WMA algorithm. The cleaning effectiveness is measured by the *mean square error* between the cleaned and true data. In the simulation, we test both strategies of computing the prediction range: the KL filter and the regression approach. Figure 4 plots errors of the raw data and smoothed data from WMA. The raw data are random samples with a lot of noise, and therefore have high errors. As shown in Figure 4, the WMA algorithm can greatly reduce the random error contained in the raw data. We also find that the performance of the two prediction approaches, KL filter and regression, is almost the same.

4.2 Response Time

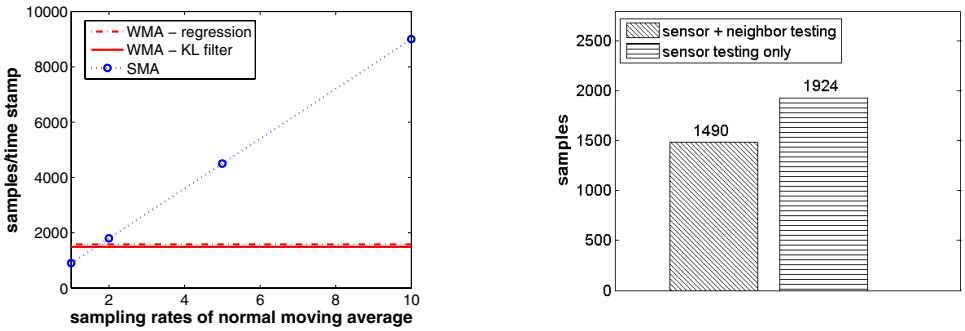
In this set of experiments, we compare the response time of WMA with that of SMA (i.e. *simple moving average*). The SMA approach in our evaluation is designed to have a constant window size $w = 10$, that is, we average 10 samples each time. A sensor can change its sampling rate, and thus sample multiple times at one timestamp. For example a sampling rate 2 in our simulation indicates that each sensor draws two samples at each timestamp. In that case, a window with size $w = 10$ lasts for five timestamps. With the SMA method, we want to remove noise and meanwhile capture the data change quickly by actively adapting the sampling rates.

We evaluate both WMA and SMA algorithms in terms of their sampling efficiency and response time to data changes. A change in the simulation is defined as a point that first exceeds $\overline{v} + \triangle t$, where $\overline{v}$ is the average of 200 previous values in the time series and $\triangle t$ is a pre-defined threshold. Figure 5-A plots the distribution of the response time. The WMA algorithm can respond to most of the changes in real-time, while SMA with sampling rate 2 shows a bell curve centering at a delay of $t = 4$ timestamps. In order to quickly



A. Distribution of response time B. Average response time

Figure 5. Response time to the change point



A. Sampling cost B. Neighobor testing

Figure 6. Efficiency

respond to the change, SMA needs to use a higher sampling rate for *all* sensors at *all* timestamps. Figure 5-B illustrates that higher sampling rates can reduce the response time. The WMA algorithm can automatically increase the sampling rate at the point where changes occur. It can use a low average sampling rate to achieve short response time.

4.3 Energy Efficiency

The use of the adaptive sampling and weighted averaging is to save the sampling cost. Figure 6-A illustrates that the WMA algorithm uses fewer samples compared with SMA most of the time. In contrast to SMA with rate 2 that consumes a similar number of samples to WMA, WMA needs only about half the response time (see Figure 5-B). The use of the neighboring test can further reduce the sampling cost, as illustrated in Figure 6-B. The figure plots the number of samples required in each timestamp averaged from 250 continuous timestamps. We can see that the neighboring test can save about 18% of the sampling cost.

5 Related Works

Sensor networks have been widely used in various applications [11, 13]. Sensor readings are often noisy and error prone due to the low quality of sensors and random effects of the environment [4, 14]. Many works have been proposed to clean noisy data before they are used for answering queries. Based on the location where data are cleaned, these proposals can be sink-based (base station) or sensor-based. For the sink-based methods, quite a few works have been done. Mukhopadhyay et al. [12] proposed a model-based approach to correct the transient errors of sensor data. Specifically, an off-line process selects the type and order of the models first, and then, based on the collected sampling sensor data, the parameters of this model are estimated to correct the data. To reduce the uncertainty associated with the data, Elnahrawy and Nath [4] proposed a Bayesian approach, which combines prior knowledge of the true sensor readings, the noise characteristics of this sensor and the observed noisy readings. However, it is often not possible or at least difficult to know the noise characteristics of a sensor. Jeffrey et al. [7] proposed a general framework for building sensor data cleaning infrastructures in pervasive applications. The main idea is to utilize the spatio-temporal correlation among the sensor data to recover lost readings and remove outliers. Subramaniam et al. [15] proposed a data cleaning approach with an online distribution estimation model and a hierarchical network structure. Kalman filter, as a stochastic, recursive data filter, has been used to

filter noise data from the senor [6]. Other than Kalman filter, some complicated approaches using multiple sensor fusion have also been proposed to remove noise, such as [16]. With respect to sensor-based data cleaning approaches, not much has been done yet. Branch et al. [2] proposed a distributed in-network outlier detection method based on the changing rate of the value and Zhuang et al. [20] detected outliers within a sensor by utilizing the spatio-temporal correlation among the sensor data.

Compared to previous works, our work is a hybrid cleaning approach, which cleans the data at the sink as well as within the sensor. We use the weighted moving average to clean the data at the sink and use spatio-temporal correlation to verify the confidence of changing data within the sensor.

6 Conclusions

In this paper, we have proposed a *weighted moving average* algorithm (WMA) to clean sensor data. Moving average is a popular approach for data cleaning [7, 5]. An SMA with higher sampling rate can improve the averaging results by drawing more samples at each timestamp. However, this is costly for sensors with limited battery power. On the other hand, when the sampling rate of SMA is low, the moving average responds slowly to changes in the data. To overcome this problem, we present a WMA which adds confidence weight to important values in the moving average, such as a change point, a pike and so on. The important values are detected by a prediction range computed from the Kalman filter or linear regression. We recursively draw samples and test whether the data is in or out of the prediction range. A confidence value is then reported to the sink, through which weights are computed for WMA.

The WMA algorithm can effectively remove random noise in the noisy data. Compared with SMA, WMA uses fewer samples to achieve a quicker response time to the data change. We conduct comprehensive simulations and show the effectiveness of the WMA algorithm and its energy efficiency.

7 Acknowledgement

Funding for this work was provided by the Hong Kong RGC grants DAG05/06.EG03, the NSFC Key Project Grant No. 60533110, National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303000, and the US NSF grant ISI-0415023.

References

[1] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 2000.

[2] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta. In-network outlier detection in wireless sensor networks. In *Proc. of ICDCS*, 2006.

[3] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proc. of SIGMOD*, 2005.

[4] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *Proc. of MSWiM*, 2003.

[5] J. Hellerstein and W. Hong and S. Madden and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *Proc. of IPSN*, 2003.

[6] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *Proc. of SIGMOD*, 2004.

[7] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative support for sensor data cleaning. In *Proc. of PerCom*, 2006.

[8] M. Singh and V. K. Prasanna. System level energy tradeoffs for collab-orative computation in wireless networks. In *Proc. of ICC*, 2002.

[9] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of OSDI*, 2002.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of SIGMOD*, 2003.

[11] Mo Li and Yunhao Liu. Underground structure monitoring with wireless sensor networks. In *Proc. of IPSN*, 2007.

[12] S. Mukhopadhyay, D. Panigrahi, and S. Dey. Model based error correction for wireless sensor networks. In *Proc. of SECON*, 2004.

[13] L. M. Ni, Y. Liu, Y. C. Lau, and A. Patil. Landmarc: Indoor location sensing using active rfid. *ACM Wireless Networks*, 2004.

[14] D. Niculescu and B. Nath. Error characteristics of ad hoc positioning systems (aps). In *Proc. of MobiHoc*, 2004.

[15] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proc. of VLDB*, 2006.

[16] G. Wu, Y. Wu, L. Jiao, Y.-F. Wang, and E. Y. Chang. Multi-camera spatio-temporal fusion and biased sequence-data learning for security surveillance. In *Proc. of ACM MM*, 2003.

[17] W. Xue, Q. Luo, L. Chen, and Y. Liu. Contour map matching for event detection in sensor networks. In *Proc. of SIGMOD*, 2006.

[18] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proc. of CIDR*, 2003.

[19] Zarchan Paul and Musoff Howard. Fundamentals of kalman filtering: A practical approach. In *AAAI*, 2007.

[20] Y. Zhuang and L. Chen. In-network outlier cleaning for data collection in sensor networks. In *Proc. of CleanDB*, 2006.