Seventh International Workshop on Data Management for Sensor Networks (DMSN'10)

September 13, 2010

Singapore

in conjunction with the

36th International Conference on Very Large Data Bases

Editors: Demetris Zeinalipour
Wang-Chien Lee

Sponsored by:

Foreword

It is our great pleasure to welcome you all to the 7th International Workshop on Data Management for Sensor Networks (DMSN'10), which takes place in Singapore on September 13, 2010. The annual DMSN workshop is a leading international forum that covers all important aspects of sensor data management, including data acquisition, processing, and storage in remote wireless networks; the handling of uncertain sensor data; and the management of heterogeneous and sometimes sensitive sensor data in databases. It brings together a wide range of researchers, practitioners, and users to explore and share scientific and industrial challenges that arise in the aforementioned contexts. We hope you find the workshop academically stimulating and the location interesting and enjoyable.

One of our main objectives was to bring forward an exciting research program, spanning both predominant and emerging fields in data management for sensor networks. DMSN'10 received 10 submissions for research papers, out of those we accepted only 6 papers. The accepted papers were thematically organized in the following categories: Data Provenance, Query Processing, Mobile Sensor Networks and Outlier Detection in Sensor Networks. In addition to research contributions, DMSN'10 features an exciting keynote talk by Prof. Kian-Lee Tan (National University of Singapore, Singapore), with title: *"What's NExT? Sensor + Cloud!?"* Finally, the program also features a panel discussion with title *"Future Directions in Sensor Data Management: A Panel Discussion"*, with panelists Dr. Yanlei Diao (University of Massachusetts Amherst, USA), Prof. Le Gruenwald (National Science Foundation, USA), Prof. Christian S. Jensen (Aarhus University) and Prof. Kian-Lee Tan (National University of Singapore, Singapore)

Besides authors that provided the content of the program, several other people have contributed to the successful organization of DMSN'10. In particular, we would like to thank our technically and geographically diverse Technical Program Committee (TPC), which enabled us to make high quality decisions. Our TPC comprised of 32 members that spanned the following continents: North America (50%), Europe (31%) and Asia (19%). Our TPC board came from both Academia (87%) and Industrial Research Labs (13%). We owe our sincere gratitude to all of these members for their excellent work in reviewing the papers and providing valuable feedback under a tight schedule. Every paper was reviewed at least by 3 TPC members. We would like to thank Microsoft for granting us permission to use the Microsoft Conference Management System (CMT) and the entire CMT support team, for their help in setting up and managing the online review process. The latest features in CMT made it extremely easy to cope with virtually all aspects of the paper evaluation process.

Our special thanks also go to the general chairs of DMSN'10 Mario Nascimento (University of Alberta, Canada) and Nesime Tatbul (ETH Zurich, Switzerland) for their frequent advice that guided us through many of the questions and concerns that arose along the way. We are also grateful to our publicity chair Olga Papaemmanouil (Brandeis University, USA) for setting up and maintaining the DMSN'10 website but also for her timely dissemination activities. Finally, we would like to thank the DMSN'10 Steering Committee: Yanlei Diao (University of Massachusetts Amherst, USA), Christian S. Jensen (Aarhus University, Denmark), Alexandros Labrinidis (University of Pittsburgh, USA), Samuel R. Madden (Massachusetts Institute of Technology, USA); and the VLDB organization, in particular the VLDB workshop chairs: Amol Deshpande (University of Maryland, USA), Zachary G. Ives (University of Pennsylvania, USA) and Anthony Kum Hoe Tung (National University of Singapore, Singapore) as well as the VLDB proceeding chairs: Yi Chen (Arizona State University, USA) and Y.C. Tay (National University of Singapore, Singapore).

Despite the economically hard times we were very fortunate to receive a sponsorship from CONET (EU's Cooperating Objects Network Of Excellence), which deals with research in the areas of embedded systems, pervasive computing and wireless sensor networks.

Last and definitely not the least, we want to thank all authors who submitted their work to DMSN'10, our panelists and all of you participating at this great workshop.

We sincerely hope you enjoy the workshop, VLDB, and Singapore!

Demetris Zeinalipour
DMSN'10 PC Co-Chair
Department. of Computer Science
University of Cyprus
CY-1678 Nicosia, Cyprus
(dzeina@cs.ucy.ac.cy)

Wang-Chien Lee
DMSN'10 PC Co-Chair
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802, USA
(wlee@cse.psu.edu)

DMSN 2010 Workshop Organization

General Chairs:
Mario A. Nascimento (*University of Alberta, Canada*)
Nesime Tatbul (*ETH Zurich, Switzerland*)

Program Committee Chairs:
Wang-Chien Lee (*Pennsylvania State University, USA*)
Demetris Zeinalipour (*University of Cyprus, Cyprus*)

Steering Committee:
Yanlei Diao (*University of Massachusetts Amherst, USA*)
Christian S. Jensen (*Aarhus University, Denmark*)
Alexandros Labrinidis (*University of Pittsburgh, USA*)
Samuel R. Madden (*Massachusetts Institute of Technology, USA*)

Program Committee:
Karl Aberer (*EPF Lausanne, Switzerland*)
Magdalena Balazinska (*University of Washington, USA*)
Erik Buchmann (*Karlsruhe Institute of Technology, Germany*)
Ugur Cetintemel (*Brown University, USA*)
Lei Chen (*Hong Kong University of Science and Technology, Hong Kong*)
Panos K. Chrysanthis (*University of Pittsburgh, USA*)
Yanlei Diao (*University of Massachusetts Amherst, USA*)
Alvaro A.A. Fernandes (*University of Manchester, UK*)
Lin Guo (*Hong Kong University of Science and Technology, Hong Kong*)
Takahiro Hara (*Osaka University, Japan*)
Wei Hong (*Arch Rock Corporation, USA*)
Christian S. Jensen (*Aarhus University, Denmark*)
Vana Kalogeraki (*Athens University of Economics and Business, Greece*)
Yannis Kotidis (*Athens University of Economics and Business, Greece*)
Philip Levis (*Stanford University, USA*)
Samuel R. Madden (*Massachusetts Institute of Technology, USA*)
Sebastian Michel (*Saarland University, Germany*)
Gail Mitchell (*BBN, USA*)
Mohamed Mokbel (*University of Minnesota, USA*)
Rene Muller (*ETH Zurich, Switzerland*)
Suman Nath (*Microsoft Research Redmond, USA*)
Ioanis Nikolaidis (*University of Alberta, Canada*)
Olga Papaemmanouil (*Brandeis University, USA*)
Kai-Uwe Sattler (*TU Ilmenau, Germany*)
Adam Silberstein (*Yahoo! Research, USA*)
Kian-Lee Tan (*National University of Singapore, Singapore*)
Xueyan Tang (*Nanyang Technological University, Singapore*)
Nesime Tatbul (*ETH Zurich, Switzerland*)
Goce Trajcevski (*Northwestern University, USA*)
Matt Welsh (*Harvard University, USA*)
Jianliang Xu (*Hong Kong Baptist University, Hong Kong*)
Jun Yang (*Duke University, USA*)

Table of Contents

What's NExT? Sensor + Cloud!?

Kian-Lee Tan
National University of Singapore, Singapore
tankl@comp.nus.edu.sg

ABSTRACT

Today, we are witnessing a number of interesting phenomena. First, there is an increasing adoption of sensing technologies (e.g., RFID, cameras, mobile phones) in many industries. Second, the internet has become a source of real-time information (e.g., through blogs, social networks, live forums) for events happening around us. In fact, we can consider these sources as "sensors". Finally, Cloud computing has emerged as an attractive solution for dealing with the "Big Data" revolution. By combining data obtained from sensors with that from the internet, we can potentially create a demand for resources that can be appropriately met by the cloud. This talk will discuss some application scenarios, challenges and opportunties for the communities. Our goal is to exploit these technolgies for smart living.

Kian-Lee Tan is a Professor of Computer Science at the School of Computing, National University of Singapore (NUS). He received his Ph.D. in computer science in 1994 from NUS. His current research interests include multimedia information retrieval, query processing and optimization in multiprocessor and distributed systems, database performance, and database security. He has published numerous papers in conferences such as SIGMOD, VLDB, ICDE and EDBT, and journals such as TODS, TKDE, and VLDBJ. Kian-Lee is a member of ACM.

DMSN '10, September 13, 2010, Singapore

Provenance-based Trustworthiness Assessment in Sensor Networks

Hyo-Sang Lim
Department of Computer Science,
Purdue University, USA
hslim@cs.purdue.edu

Yang-Sae Moon
Department of Computer Science,
Kangwon National University, South Korea
ysmoon@kangwon.ac.kr

Elisa Bertino
Department of Computer Science,
Purdue University, USA
bertino@cs.purdue.edu

ABSTRACT

As sensor networks are being increasingly deployed in decision-making infrastructures such as battlefield monitoring systems and SCADA(Supervisory Control and Data Acquisition) systems, making decision makers aware of the trustworthiness of the collected data is a crucial. To address this problem, we propose a systematic method for assessing the trustworthiness of data items. Our approach uses the data provenance as well as their values in computing trust scores, that is, quantitative measures of trustworthiness. To obtain trust scores, we propose a cyclic framework which well reflects the *inter-dependency* property: the trust score of the data affects the trust score of the network nodes that created and manipulated the data, and vice-versa. The trust scores of data items are computed from their *value similarity* and *provenance similarity*. The value similarity comes from the principle that "the more similar values for the same event, the higher the trust scores". The provenance similarity is based on the principle that "the more different data provenances with similar values, the higher the trust scores". Experimental results show that our approach provides a practical solution for trustworthiness assessment in sensor networks.

1. INTRODUCTION

Advances in hardware and network technologies enable the development of large-scale sensor networks in a large variety of novel applications, like supervisory systems, e-health, and e-surveillance. In near future, sensor networks will be deployed everywhere and consist of thousands to millions of tiny sensor nodes as we can see from the Smart Dust project [7] which aims to create grain-of-sand sized sensors. In such new environments, sensor networks collect large amounts of data that can convey important information for critical decision making. Thus, being able to assess the trustworthiness of the collected data and making decision makers aware of the trustworthiness of these data become crucial.

A possible approach to this problem is to associate a trust score with each data item. Such score provides an indication about the trustworthiness of the data item and can be used for data comparison or ranking. For example, even though the meaning of absolute scores varies depending on the application or parameter settings, if a data item has the highest trust score in a data set, then we can say that the data item is the most trustworthy compared with the other data items in the set. Also, as indicators about data trustworthiness, trust scores can be used together with other factors (e.g., information about contexts and situations, past data history) for deciding about the use of data items. A critical element in solutions to assign trust score to data is the method for computing the data trust scores. The goal of this paper is to develop such a method for data collected in sensor networks. Our approach is based on the concept of provenance, as provenance gives important evidence about the origin of the data, that is, where and how the data is generated. Provenance provides knowledge about how the data came to be in its current state - where the data originated, how it was generated, and the operations it has undergone since its creation.

Our method is based on the principle that the more trustworthy data a source provides, the more trusted the source is considered. There is thus an interdependency between network nodes and data items with respect to the assessment of their trust scores, i.e., the trust score of the data affects the trust score of the network nodes that created and manipulated the data, and vice-versa. To reflect such interdependency in computing trust scores, we propose a cyclic framework that generates: (1) trust scores of data items from those of network nodes and (2) trust scores of network nodes from those of data items. Trust scores are gradually evolved in our cyclic framework.

Our framework works as follows. Trust scores are initially computed based on the values and provenance of data items; we refer to these trust scores as *implicit trust scores*. To obtain these trust scores, we use two types of similarity functions: *value similarity* inferred from data values, and *provenance similarity* inferred from data provenances. Value similarity is based on the principle that the more data items referring to the same real-world event have similar values, the higher the trust scores of these items are. We thus propose a systematic approach for computing trust scores based on value similarity under the distribution of collected data. Provenance similarity is based on the observation that different provenances of similar data values may increase the trustworthiness of data items. In other words, different provenances provide more independent data items. In the paper we thus present a formal model for computing the provenance similarity and integrating it into the data similarity.

We have implemented the cyclic framework for computing trust scores. Through extensive experiments, we first show that our method works correctly in sensor networks and the cyclic framework gradually evolves trust scores by reflecting changes in sensing

DMSN'10, September 13, 2010, Singapore
.

value changes. These experimental results show that our approach provides a practical solution for trustworthiness assessment in sensor networks.

The rest of the paper is organized as follows. Section 2 models sensor networks and data provenances. Section 3 proposes the cyclic framework for generating trust scores of data items and network nodes based on their values and provenance. Section 4 reports the experimental results. We finally summarize and conclude the paper in Section 5.

2. DATA PROVENANCE AND ITS REPRESENTATION

Networks are usually modeled as graphs. We thus model the physical (sensor) network as a graph of $G(N, E)$, where the set of nodes, N, and the set of edges, E, are defined as follows:

- $N = \{n_i \mid n_i$ is a network node of whose identifier is $i.\}$: a set of network nodes
- $E = \{e_{i,j} \mid e_{i,j}$ is an edge connecting nodes n_i and $n_j.\}$: a set of edges connecting nodes

Figure 1 (a) shows an example of a sensor network.

Regarding the network nodes in N, we categorize them into three types according to their roles.

Definition 1. *A terminal node generates a data item and sends it to one or more intermediate or server nodes. An intermediate node receives data items from one or more terminal or intermediate nodes, and it passes them to intermediate or server nodes; it may also generate an aggregated data item from the received data items and send the aggregated item to intermediate or server nodes. A server node receives data items and evaluates the user queries based on those items.* □

Without loss of generality, we assume that in G there is only one server node, denoted by n_s. To simplify the presentation, we only consider a single numeric value as a data item. However, we can easily extend our solution to multiple attributes by separately assigning independent scores to each attribute or by exploiting multi-attribute distributions. In this paper, we also only focus on handling selection and aggregation which are the most used operations in sensor networks. We will explore additional other operations in our future work.

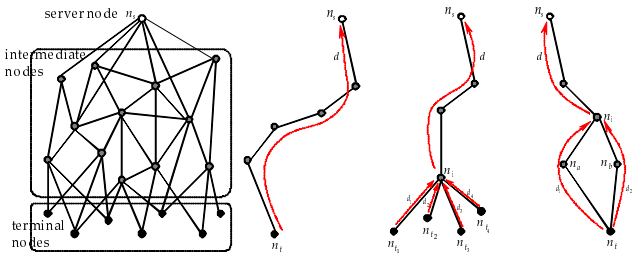


Figure 1: A physical sensor network and data provenance examples.

We now define the *provenance* of a data item d, denoted as p_d. The provenance p_d records where and how the data item d was generated and how it was passed to the server n_s.

Definition 2. *The provenance p_d of a data item d is a rooted tree satisfying the following properties: (1) p_d is a subgraph of the physical sensor network $G(N, E)$; (2) the root node of p_d is the server node n_s; (3) for two nodes n_i and n_j of p_d, n_i is a child of n_j if and only if n_i has passed the data item d to n_j.* □

We categorize intermediate nodes in data provenance into two types based on their operations. The *simple nodes* are internal node having only one child. The simple nodes simply pass data items from their children to their parents. Simple nodes are typically used in ad-hoc sensor networks to relay data items to a server in order to address the insufficient capability of data transmission. The *aggregate node* are internal nodes having two or more children nodes. They receive multiple data items from their multiple children, generate aggregated data items, and pass them to their parents.

Figures 1 (b) and 1 (c) show some examples of the two different data provenances. As shown in the figures, data provenances are subgraphs of the physical sensor network of Figure 1 (a), and they are trees rooted at the server node n_s. In Figure 1 (b) every intermediate node in the provenance p_d is a simple node, which means that the data item d is generated in a terminal node n_t and simply passed to the server n_s. We call this type provenance a *simple provenance*, which can be represented as a simple path. On the other hand, in Figure 1 (c) an internal node n_i is an aggregate node, which means that n_i generates a new data item d by aggregating multiple data items $d_1, \ldots, d_4$ from $n_{t_1}, \ldots, n_{t_4}$ and passes d to the server n_s. We call this type provenance an *aggregate provenance*, which is represented as a tree rather than a simple path.

According to Definition 2, a data provenance should be a tree. However, there could be cycles and thus the provenance is not a tree such as the example in Figure 1 (d). We do not consider this case because of two reasons. First, it rarely occurs in real environments. Second, tree similarity can be computed in $O(n^3 \log n)$ [6]; in contrast, computing graph similarity is known as an NP-hard problem [5] (refer to Section 3 for details). We note that basically there is no much difference between tree-shaped and graph-shaped provenances (Only minor changes are required to support graph-based provenance).

3. PROVENANCE-BASED TRUST SCORE COMPUTATION

In this section, we present our cyclic framework for computing trust scores of data items and network nodes.

3.1 Cyclic Framework for Incremental Update of Trust Scores

We derive our cyclic framework based on the *interdependency* [1, 3] between data items and their related network nodes. The interdependency means that the trust scores of data items affect the trust scores of network nodes, and similarly the trust scores of network nodes affect those of the data items. In addition, the trust scores need to be continuously evolved in the stream environment since new data items continuously arrive to the server. Thus, a cyclic framework is adequate to reflect the interdependency and continuous evolution properties. Figure 2 shows the cyclic framework according to which the trust scores of data items and the trust scores of network nodes are continuously updated. Note that we consider a sensor network where there are multiple sensors for monitoring an event (i.e., we can get multiple independent observations for an event), and thus trust scores are computed for the data items concerning the same event in a given streaming window.

As shown in Figure 2, we maintain three different types of trust scores, *current*, *intermediate*, and *next trust scores* to reflect the in-

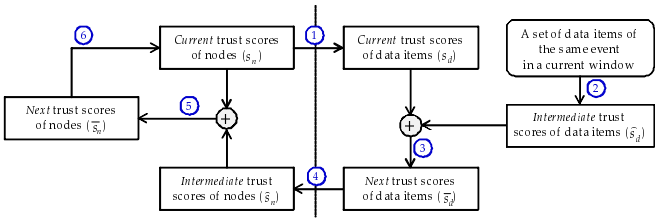


Figure 2: A cyclic framework of computing trust scores of data items and network nodes.

terdependency and continuous evolution properties in the computation of the trust scores. The trust scores of data items and network nodes well reflect those properties as many as cycles are repeated. We explain the detailed computation process for the trust scores of data items and network nodes in Section 3.3 and 3.2, respectively.

It is important to note that these scores are mainly indicators, to be used for example for comparison purpose. For example, let s_1 and s_2 be trust scores of data d_1 and d_2. If $s_1 > s_2$, d_1 is more trustworthy than d_2. The meaning of absolute scores varies depending from the specific applications or parameter values.

3.2 Computing Trust Scores of Network Nodes

For a network node n whose current score is s_n, we are about to compute its next score $\bar{s}_n$. In more detail, the trust score of n was computed as s_n in the previous cycle, and we now recompute the trust score as $\bar{s}_n$ using a set of recent data items in a streaming window in order to determine how the trust score has to evolve in a new cycle. We compute the next score based on the following two principles: 1) the intermediate score $\hat{s}_n$ reflects the trust scores of its related data items based on the interdependency property; 2) the next score $\bar{s}_n$ reflects its current and intermediate scores s_n and $\hat{s}_n$ to gradually evolve the trust scores of network nodes.

We now show how to compute $\hat{s}_n$ and $\bar{s}_n$. First, let D_n be a set of data items that are issued from or passed through n in the given streaming window. That is, all data items in D_n are identified as related to the same event, and they are issued from or passed through the network node n. We adopt the idea that "higher scores for data items ($\in D_n$) result in higher scores for their related node (n)" [1, 2]. Thus, $\hat{s}_n$ is simply computed as the average of $\bar{s}_d$'s ($d \in D_n$), which are the next trust scores of data items in D_n:

$$\hat{s}_n = \frac{\sum_{d \in D_n} \bar{s}_d}{|D_n|} \tag{1}$$

In Eq. (1) we note that the trust score of a network node is determined by the trust scores of its related data items, and this satisfies the first principle, that is, interdependency property. Also, the next score $\bar{s}_n$ is computed as a weighted-sum of s_n and $\hat{s}_n$:

$$\bar{s}_n = c_n s_n + (1 - c_n)\hat{s}_n,$$
$$\text{where } c_n \text{ is a given constant of } 0 \leq c_n \leq 1. \tag{2}$$

In Eq. (2) we note that this satisfies the second principle, i.e., the consideration of current and intermediate scores.

The constant c_n in Eq. (2) represents how fast the trust score is evolved as the cycle is repeated. For example, if c_n has a larger value, especially if $c_n > \frac{1}{2}$, we consider s_n to be more important than $\hat{s}_n$, and this means that the previously accumulated historic score (s_n) is more important than the latest trust score ($\hat{s}_n$) recently computed from data items in D_n. On the other hand, if c_n has a smaller value, especially if $c_n < \frac{1}{2}$, we consider the latest score $\hat{s}_n$ to be more important than the historic score s_n. In summary, if c_n is large, the trust score will be evolved slowly; in contrast, if c_n is small, the trust score will be evolved fast.[1]

3.3 Computing Trust Scores of Data Items

Basically we compute the trust score of a data item d using its value v_d and provenance p_d. In this paper, we model the distributions of data items in the same event as a *normal (Gaussian) distribution*. In more detail, for data items in a set D of data items related to the same event, we model the distribution of D as a probability density function $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, where x is the attribute value v_d of a data item $d\,(\in D)$, and μ and σ^2 are mean and variance of D respectively. We use the normal distribution since it well reflects natural phenomena. Especially, values sensed for one purpose in general follow a normal distribution [4, 8], and thus this distribution is a reasonable choice for modeling streaming data items in sensor networks. However, we note that the normal distribution assumption is not a limit of our solution. We just use the distribution for estimating similarities among data values in the trust score computation. We can adopt other distributions, histograms, or correlation information with simple changes to the data similarity models.

3.3.1 *Current trust score* s_d

For a data item d, we first compute its current score s_d based on current scores of network nodes in its provenance p_d (see ① in Figure 2). This process reflects the interdependency property because we use the trust scores of network nodes for those of data items. In Section 2 we have explained the two different types of provenance: one is the simple provenance with a path structure; the other is the aggregate provenance with a tree structure. According to this classification, in what follows we first show how to compute the current score s_d for the simple provenance and then extend it for the aggregate provenance.

In the case of the simple provenance (like in Figure 1 (b)), we can represent it as $p_d = (n_1, n_2, \ldots, n_k = n_s)$, that is, as a sequence of network nodes that d passes through. In this case, we determine the current score s_d on the minimum among the scores of all nodes in p_d. This is based on an intuition that, if a data item passes through network nodes in a sequential order, its trust score might be dominated by the worst node with the smallest trust score[2]. That is, we compute s_d as follows:

$$s_d = \min\{s_{n_i} |\, n_i \in p_d\} \tag{3}$$

If a data item d has an aggregate provenance p_d, we need to consider the tree structure (like in Figure 1 (c)) to compute its current score s_d. For an aggregate node, we first obtain a representative score by aggregating the current scores of its child nodes and then use this aggregate score as the current score of the child nodes. We use an average score of child nodes as their aggregated score[3]. By recursively executing this aggregation process, we simplify a tree into a simple path of aggregated scores, and we finally compute the current score s_d by taking their minimum score as in Eq. (3).

Algorithm 1 shows a recursive solution for computing the current score s_d from its provenance p_d, which can be either a simple or

[1] In the experiment we set $c_n = \frac{1}{2}$ to equally reflect the importance of s_n and $\hat{s}_n$, and we assume that the first value of s_n is set to 1.

[2] We can also use an average score or weighted average score of network nodes to compute the current score. In this case, we obtain the score by simply changing the minimum function to the average or weighted average function in Eq. (3).

[3] According to the aggregate operation applied to the aggregate node, we can use different methods. That is, for `AVG` we can use an average of children, but for `MIN` or `MAX` we can use a specific score of a network node that produces a resulting minimum or maximum value. An aggregation itself, however, represents multiple nodes, and we thus use the average score of child nodes as their representative score.

aggregate provenance. To obtain the current score s_d of a data i
d, we simply call *CompCurrentScore*(n_s) where n_s is the root n
of p_d.

Algorithm 1 *CompCurrentScore* (n_i: a tree node in p_d)

```
1: if n_i is a simple node (i.e., n_i has only one child) then
2:    Let n_j be the child node of n_i;  // an edge e_{i,j} connects two nodes.
3:    return MIN(s_{n_i}, CompCurrentScore(n_j));
4: else if n_i is an aggregate node with k children then
5:    Let n_{j_1}, ..., n_{j_k} be k child nodes of n_i;
6:    return MIN(s_{n_i}, AVG(CompCurrentScore(n_{j_1}), ...,
7:                  CompCurrentScore(n_{j_k})));
8: else  // n_i is a leaf node.
9:    return s_{n_i};
10: end-if
```

3.3.2 *Intermediate trust score* $\hat{s}_d$

An intermediate trust score $\hat{s}_d$ of a data item d is computed fr
the latest set of data items of the same event with d in the cur
streaming window (see ② in Figure 2). Let D be the set of
items in the same event with d. In general, if set D changes,
a new item is added to D or an item is deleted from D, we rec
pute the trust scores of the data items in D. We obtain $\hat{s}_d$ thro
the initial and adjusting steps. In the initial step, we use the *va*
similarity of data items in computing an initial value of $\hat{s}_d$. In
adjusting step, we use the *provenance similarity* to adjust the ini
value of $\hat{s}_d$ by considering provenances of data items.

(1) Initial score of $\hat{s}_d$ based on value similarity

Based on our normal distribution model, we observe that, for a
D of a single event, its mean μ is the most representative va
that well reflects the value similarity. This is because the mea
determined by the majority values, and obviously those majo
values are similar to the mean in the normal distribution. Thus,
conclude that the mean has the highest trust score; if the valu
a data item is close to the mean, its trust score is relatively hig
the value is far from the mean, its trust score is relatively low.

Based on those observations, we propose a method to compute the intermediate score s_d in the initial step. In obtaining $\hat{s}_d$, we assume that $v_d \geq \mu$. We can easily extend our method to the case of $v_d \leq \mu$, and we thus omit that case for simplicity.

As intermediate score $\hat{s}_d$, we use the cumulative probability of the normal distribution. In this method, we use "$1 -$ the amount of how far v_d is from the mean" as the initial score of $\hat{s}_d$, and here "the amount of how far v_d is from the mean" can be thought as the cumulative probability of v_d. Thus, as in Eq. (4), we obtain the initial $\hat{s}_d$ as the integral area of $f(x)$.

$$\hat{s}_d = 2\left(0.5 - \int_{\mu}^{v_d} f(x)\,dx\right) = 1 - \int_{2\mu - v_d}^{v_d} f(x)\,dx = 2\int_{v_d}^{\infty} f(x)\,dx \quad (4)$$

Figure 3 shows how to compute the integral area for the initial intermediate score s_d. In the figure, the shaded area represents the initial score of $\hat{s}_d$, which is obviously in (0,1]. Here, the score $\hat{s}_d$ increases as v_d is close to μ.

According to our data similarity model, if a sensor suddenly generates a data value which is different from the mean, this data value will initially receive a low trust score. However, in this case, the system provides users with an explanation about why the trust score is so low. There could be two possible reasons for the trust score of a data generated by a sensor to be low: 1) the observation by the sensor is quite different from the other observations of the same

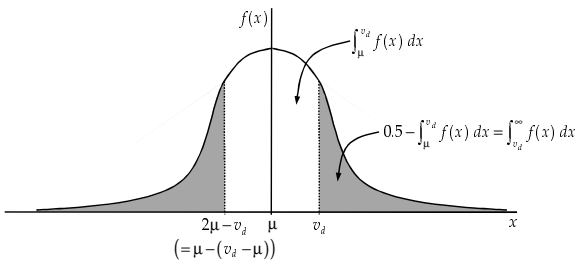


Figure 3: Computing the intermediate score of $\hat{s}_d$.

event; 2) the observation is currently the only observation for the event. In the former case, users can safely conclude that the data value is wrong. In the latter case, users can take different actions. They can just wait for the arrival of new data concerning this event. Or they can activate additional sensors (for example we may assume that not all sensors are always activated in order to save energy), that in turn will result in more data to be generated. If the initial observation is actually true, other sensors will send similar observations shortly, and then, the cyclic framework will automatically reflect this situation by increasing the trust scores of the initial data value.

(2) Adjusted score of $\hat{s}_d$ based on provenance similarity

We need to adjust the intermediate score $\hat{s}_d$ by reflecting the provenance similarity of data items. To achieve this, we let a set of provenances in D be P and the similarity function between two data provenances p_i, p_j ($\in P$) be $sim(p_i, p_j)$[4]. Here, the similarity function $sim(p_i, p_j)$ returns a similarity value in $[0, 1]$, and can be computed from the tree or graph similarity [5, 6]. Computing graph similarity, however, is known to be an NP-hard problem [5], and we thus use the tree similarity [6], which is an edit distance-based similarity measure.

Our approach to take into account provenance similarity in computing the intermediate score $\hat{s}_d$ is based on some intuitive observations. In the following, notation '$\sim$' means "is similar to", and notation $\nsim$ means "is not similar to." Given two data items $d, t \in D$, their values v_d, v_t, and their provenances $p_d, p_t \in P$,

- if $p_d \sim p_t$ and $v_d \sim v_t$, the provenance similarity makes a *small positive* effect on $\hat{s}_d$;
- if $p_d \sim p_t$ and $v_d \nsim v_t$, the provenance similarity makes a *large negative* effect on $\hat{s}_d$;
- if $p_d \nsim p_t$ and $v_d \sim v_t$, the provenance similarity makes a *large positive* effect on $\hat{s}_d$;
- if $p_d \nsim p_t$ and $v_d \nsim v_t$, the provenance similarity makes a *small positive* effect on $\hat{s}_d$;

Based on the above observations, we introduce a measure of *adjustable similarity* to reflect the provenance similarity in adjusting $\hat{s}_d$. Given two data items d, t ($\in D$), we first define the adjustable similarity between d and t, denoted by $\rho_{d,t}$, as follows:

$$\rho_{d,t} = \begin{cases} 1 - sim(p_d, p_t), & \text{if } dist(v_d, v_t) < \delta_1 \text{; // positive effect} \\ -sim(p_d, p_t), & \text{if } dist(v_d, v_t) > \delta_2 \text{; // negative effect} \\ 0, & \text{otherwise. // no effect} \end{cases} \quad (5)$$

In Eq. (5), $dist(v_d, v_t)$ is a distance function between v_d and v_t; δ_1 is a threshold indicating when v_d and v_t are to be treated as similar; δ_2 is a threshold indicating when v_d and v_t are to be treated as

[4] Data items in the same event may have similar provenances, so we may assume that the number of possible provenances for an event is finite and actually small. Thus, for real-time processing purposes, we can materialize all $sim(p_i, p_j)$'s in advance and maintain them in memory.

dissimilar[5]. The adjustable similarity $\rho_{d,t}$ in Eq. (5) well reflects the effect of provenance and value similarities. That is, if v_d and v_t are similar, $\rho_{d,t}$ has a positive value of "$1 - sim(p_d, p_t)$" determined by the provenance similarity; in contrast, if they are not similar, $\rho_{d,t}$ has a negative value of "$-sim(p_d, p_t)$." To consider adjustable similarities of all data items in D, we now obtain their sum ρ_d as follows:

$$\rho_d = \sum_{t \in D, t \neq d} \rho_{d,t} \tag{6}$$

We then adjust the value v_d by considering ρ_d and use the adjusted value, denoted by $\bar{v}_d$, to compute $\hat{s}_d$ instead of v_d. In more detail, we first normalize ρ_d into $[-1, 1]$ using its maximum and minimum similarities, $\rho_{\max}$ and $\rho_{\min}$. The normalized value of ρ_d, denoted by $\bar{\rho}_d$, is thus computed as follows:

$$\bar{\rho}_d = 2\frac{\rho_d - \rho_{\min}}{\rho_{\max} - \rho_{\min}} - 1, \quad \text{where } \rho_{\max} = \max\{\rho_t | \, t \in D\} \text{ and } \rho_{\min} = \min\{\rho_t | \, t \in D\} \tag{7}$$

We then adjust the data value v_d to a new value $\bar{v}_d$ as follows:

$$\bar{v}_d = \min\{v_d - \bar{\rho}_d(c_p \cdot \sigma), \mu\}, \quad \text{where } c_p \text{ is a constant greater than 0.} \tag{8}$$

Figure 4 shows how the value v_d changes to $\bar{v}_d$ based on the adjustable similarity $\bar{\rho}_d$ in the framework of a normal distribution. As shown in the figure, if $\bar{\rho}_d > 0$, i.e., if the provenance similarity makes a positive effect, v_d moves to the left in the distribution graph, i.e., the intermediate score $\hat{s}_d$ increases; in contrast, if $\bar{\rho}_d < 0$, i.e., if the provenance similarity makes a negative effect, v_d moves to the right in the graph, i.e., $\hat{s}_d$ decreases. In Eq. (8), c_p represents the important factor of provenance similarity in computing the intermediate score. That is, as c_p increases, the provenance similarity becomes more important. We use 0.2 as the default value of c_p, i.e., we move the data value v_d in $\pm 20\%$ range of the standard deviation σ.

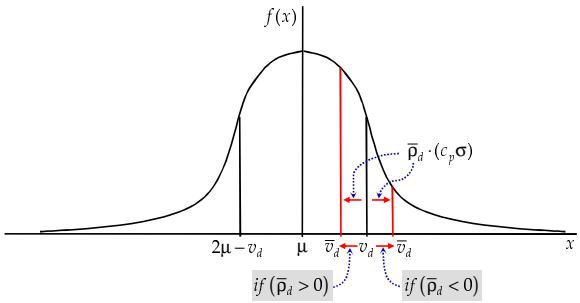


Figure 4: The effect of the provenance similarity on a data value.

By using the adjusted data value $\bar{v}_d$, we finally recompute the intermediate score $\hat{s}_d$. By simply changing v_d to $\bar{v}_d$, we can also obtain Eq. (9) from Eq. (4) in which the integral area for the intermediate score may increase or decrease by the provenance similarity.

$$\hat{s}_d = 2\int_{\bar{v}_d}^{\infty} f(x)\,dx = 1 - \int_{2\mu - \bar{v}_d}^{\bar{v}_d} f(x)\,dx \tag{9}$$

[5]In the experiment we set δ_1 and δ_1 to 20% and 80% of the average distance, respectively.

3.3.3 *Next trust score* $\bar{s}_d$

For a data item d we eventually compute its next trust score $\bar{s}_d$ by using the current score s_d and the intermediate score $\hat{s}_d$. In obtaining $\bar{s}_d$, we use s_d for the interdependency property since s_d is computed from network nodes, and we exploit $\hat{s}_d$ for the continuous evolution property since $\hat{s}_d$ is obtained from the latest set of data items. Similar to computing the next score $\bar{s}_n$ of a network node n in Eq. (2), we compute $\bar{s}_d$ as follows:

$$\bar{s}_d = c_d s_d + (1 - c_d)\hat{s}_d, \quad \text{where } c_d \text{ is a given constant of } 0 \leq c_d \leq 1. \tag{10}$$

As shown in Eq. (10), the next score $\bar{s}_d$ is gradually evolved from the current and intermediate scores s_d and $\hat{s}_d$. We also note that $\bar{s}_d$ will be used to compute the intermediate scores (i.e., $\hat{s}_n$) of network nodes in the next computation cycle (see ④ in Figure 2) for the interdependency and continuous evolution principles.

Like constant c_n used in computing s_n for a network node n in Eq. (2), constant c_d in Eq. (10) represents how fast the trust score evolves as the cycle is repeated. If c_d is large, the trust scores of data items evolve slowly; in contrast, if c_d is small, they evolve fast.[6]

In this section, instead of calibrating our model with real data sets, we present general principles for choosing parameter values (e.g., confidence ranges control the tradeoff between the number and quality of results, c_n controls how fast scores are evolved). We believe these principles can be used in most applications.

4. EXPERIMENTAL EVALUATION

In this section, we present our performance evaluation. In what follows, we first describe the experimental environment, and then present the experimental results.

4.1 Experimental Environment

The goal of our experiments is to evaluate the *efficiency* and *effectiveness* of our approach for the computation of trust scores. To evaluate the efficiency, we measure the elapsed time for processing a data item with our cyclic framework in the context of a large scale sensor network and a large number of data items. To evaluate the effectiveness, we simulate an injection of incorrect data items into the network and show that trust scores rapidly reflect this situation.

We simulate a sensor network for the experiments. For simplicity, we model our sensor network as an f-ary complete tree whose fanout and depth are f and h, respectively. We vary the values of f and h to control the size of sensor networks for assessing the scalability of our framework. We also set the number of unique events to N_{event}.

We use synthetic data that has a single attribute whose values follow a normal distribution with mean μ_i and variance ${\sigma_i}^2$ for each event i ($1 \leq i \leq N_{event}$). To generate data items, for each event, we assign N_{assign} leaf nodes of the sensor network with an interleaving factor $N_{interleave}$. This means that the data items for an event are generated at N_{assign} leaf nodes and the interval between the assigned nodes is $N_{interleave}$ (e.g., if $N_{interleave} = 0$, then N_{assign} nodes are exactly adjacent with each other). To simulate the incorrect data injection, we randomly choose an event and a node assigned for the event, and then, generate a random value.

For computing the similarity between two provenances p_i and p_j (i.e., *sim*(p_i,p_j)), we use a path edit distance defined as follows:

$$sim(p_i, p_j) = 1 - \frac{1}{h}\sum_{k=1}^{h} \frac{\text{node distance between } p_i \text{ and } p_j \text{ at the } k\text{-th level}}{\text{total number of nodes at the } k\text{-th level}}$$

[6]In the experiment we set $c_d = \frac{1}{2}$ to equally reflect the importance of s_d and $\hat{s}_d$.

Here, the node distance is defined as the number of nodes between two nodes at the same level.

All the experiments have been conducted on a PC with a 2.2GHz Core2 Duo processor and 2GB RAM running Windows/XP. The program code has been written in Java with JDK 1.6.0. Table 1 summarizes the experimental parameters and their default values. In all experiments we use the default values unless mentioned otherwise.

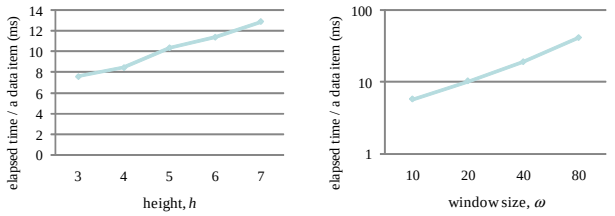
Table 1: Summary of notation.

Symbols	Definitions	Default
h	height of the sensor network	5
f	fanout of the sensor network	8
N_{event}	# of unique events	1000
N_{assign}	# of nodes assigned for an event	30
$N_{interleave}$	interleaving factor	1
ω	size of window for each event	20

As can be seen in Table 1 we only vary some application insensitive parameters. The other parameters (e.g., weights, thresholds) may be more sensitive to application contexts (e.g., data distributions, attack patterns). We will consider these parameters in the context of specific applications in our future work.

4.2 Experimental Results

(1) *Computation efficiency*: We measured the elapsed time for processing a data item. Figure 5 reports the elapsed times for different values of h's and ω's.



(a) Varying height of the sensor network (b) Varying the window size

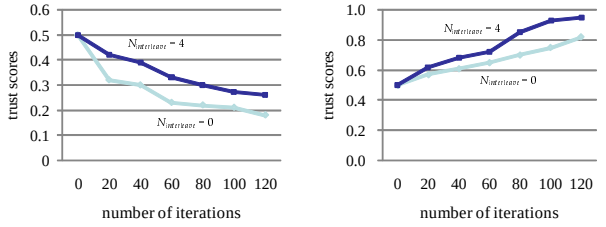
Figure 5: Elapsed times for computing trust scores.

From Figure 5 (a), we can see that the elapsed time increases as h increases. The reason is that, as h increases, the length of provenance also increases. However, the increasing rate is not high; for example, the elapsed time increases only by 9.7% as h varies from 5 to 6. The reason is that the additional operations for longer provenance linearly increase when computing the trust scores for both data items and network nodes. For the data items, only s_d and $\hat{s}_d$ are affected by the length of the provenance, i.e., an additional iteration is required to compute the weighted sum for s_d and a provenance similarity comparison for $\hat{s}_d$. For the network nodes, the computation cost increases linearly with the height (not with the total number of nodes), since we consider a very small number of network nodes related to the provenance of the new data item.

From Figure 5(b), we can see that the elapsed time increases more sharply as ω increases. The reason is that the number of similarity comparisons (not an iteration) for $\hat{s}_d$ linearly increases as ω increases. However, we can see that the performance is still adequate for handling high data input rates; for example, when ω is 80, the system can process 25 data items per second.

(2) *Effectiveness*: To assess the effectiveness of our approach, we injected incorrect data items into the sensor network, and then observed the change of trust scores of data items. Figure 6 shows the trend in trust score changes for different values of the interleaving factor $N_{interleave}$. Here, $N_{interleave}$ affects the similarity of provenances for an event, i.e., if $N_{interleave}$ increases, the provenance similarity decreases.

Figure 6 (a) shows the changes in the trust scores when incorrect data items are injected. The figure shows that the trust scores change more rapidly when $N_{interleave}$ is smaller. This trend is explained by the principle "different values with similar provenance result in a large negative effect." In contrast, Figure 6 (b) shows the changes when the correct data items are generated again. In this case, we can see that the trust scores are modified more rapidly when $N_{interleave}$ is larger. This trend is explained by the principle "similar values with different provenance result in a large positive effect."



(a) With untrustworthy data items (b) With trustworthy data items

Figure 6: Changes of trust scores for incorrect data items.

5. CONCLUSIONS

In this paper we propose a systematic method for computing and evolving the trustworthiness levels of data items/network nodes. We first introduce a cyclic framework of computing actual trust scores of data items and network nodes based on the interdependency between data and network nodes. We then provide a formal method for computing trust scores based on the *value* and *provenance similarities* of data items. Through extensive experiments, we show that our cyclic framework works well in sensor networks.

As future work, we plan to: (1) consider multiple *dependent* attributes and multi-attributes in-network operations and (2) consider other probability distributions instead of normal distributions.

Acknowledgements: The work of Elisa Bertino and Hyo-Sang Lim has been partially supported by Northrop Grumman as part of the NGIT Cybersecurity Research Consortium and by the NSF Grant N.0964294 "NeTS: Medium: Collaborative Research: A Comprehensive Approach for Data Quality and Provenance in Sensor Networks".

6. REFERENCES

[1] E. Bertino, C. Dai, H.-S. Lim, and D. Lin, "High-Assurance Integrity Techniques for Databases," In *Proc. of the 25th British Nat'l Conf. on Databases*, Cardiff, UK, pp. 244-256, July 2008.

[2] C. Dai, D. Lin, E. Bertino, and M. Kantarcioglu, "An Approach to Evaluate Data Trustworthiness Based on Data Provenance," In *Proc. of the 5th VLDB Workshop on Secure Data Management*, Auckland, New Zealand, pp. 82-98, Aug. 2008.

[3] C. Dai et al., "Query Processing Techniques for Compliance with Data Confidence Policies," In *Proc. of the 6th VLDB Workshop on Secure Data Management*, Lyon, France, pp. 49-67, 2009.

[4] E. Elnahrawy and B. Nath, "Cleaning and Querying Noisy Sensors," In *Proc. of the 2nd ACM Int'l Conf. on Wireless Sensor Networks and Applications*, San Diego, California, pp. 78-87, Sept. 2003.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, 1990.

[6] P. N. Klein, "Computing the Edit-distance between Unrooted Ordered Trees," In *Proc. of the 6th Annual European Symposium on Algorithms (ESA)*, Venice, Italy, pp 91-102, Aug. 1998.

[7] Smart Dust Project, http://robotics.eecs.berkeley.edu /∼pister/SmartDust/.

[8] M. Rabbat and R. Nowak, "Distributed Optimization in Sensor Networks," In *Proc. of the 3rd Int'l Symp. on Information Processing in Sensor Networks*, Berkeley, California, pp. 20-27, Apr. 2004.

Facilitating Fine Grained Data Provenance using Temporal Data Model

Mohammad R. Huq
University of Twente
Enschede, The Netherlands.
m.r.huq@utwente.nl

Andreas Wombacher
University of Twente
Enschede, The Netherlands.
a.wombacher@utwente.nl

Peter M. G. Apers
University of Twente
Enschede, The Netherlands.
p.m.g.apers@utwente.nl

ABSTRACT

E-science applications use fine grained data provenance to maintain the reproducibility of scientific results, i.e., for each processed data tuple, the source data used to process the tuple as well as the used approach is documented. Since most of the e-science applications perform on-line processing of sensor data using overlapping time windows, the overhead of maintaining fine grained data provenance is huge especially in longer data processing chains. This is because data items are used by many time windows. In this paper, we propose an approach to reduce storage costs for achieving fine grained data provenance by maintaining data provenance on the relation level instead on the tuple level and make the content of the used database reproducible. The approach has prototypically been implemented for streaming and manually sampled data.

Keywords

E-science applications, Sensor data, Fine grained data provenance, Temporal data model

1. INTRODUCTION

Sensors have become very common in our day-to-day lives and are used in many applications. Sensor data are acquired and processed to higher level events used in applications for decision making and process control. Events are often processed continuously in a streaming fashion to facilitate ongoing processes. In many applications it is important that the origin of processed data can be explained to understand the semantics of the event and to reproduce events. Data provenance documents the origin of data by explicating the relation of input data, algorithm, and processed data. Thus, data provenance can be used to derive event semantics. Data provenance can be defined on data relation level called coarse grained data provenance or on data tuple level called fine grained data provenance [4].

Provenance is applied on different kinds of sensor data: Streaming data is continuously produced data, while manually sampled data is a small set of data produced at a particular point in time [1, 6]. While streaming data is never updated, sampled data might be updated.

In case of fine grained data provenance, storage cost is linear with the number of sensors and processed data. Stream processing is often based on sliding windows resulting in a single data tuple contributing to many processed data. As a consequence, fine grained or tuple-based data provenance has to refer to a single tuple multiple times depending on the overlap of two subsequent sliding windows. Thus, the storage costs for tuple-based data provenance with many overlapping data in a sliding window can result in a multitude of provenance data related to the actual sensor data. The aim of this research is to provide tuple-based data provenance functionality with reduced storage costs to keep data provenance in streaming scenarios manageable.

Though the volume of manually sampled data is much less than streaming data, manually sampled data can be updated. If a piece of data is updated or deleted from the database, relation-based data provenance cannot extract the original data again. Tuple-based data provenance can solve this problem. But once there will be an update, new provenance data should be also preserved. Moreover, manually sampled data are often combined with streaming data and processed together to achieve more meaningful results. Thus this combination will end up with high volume of provenance data to be maintained compared to the actual sensor data. Therefore, low-cost tuple-based data provenance functionality should be realized in an environment where both streaming and manually sampled data are handled together.

Our proposed approach provides tuple-based data provenance with reduced storage costs by maintaining relation-based data provenance and using a temporal data model. We add timestamps to each tuple which allows us to retrieve a particular database state based on a given timestamp. Then using coarse grained provenance data, we can figure out the original tuples participated in a query to produce output tuples. The additional storage costs of these temporal attributes along with the cost of relation-based data provenance together will not exceed the storage costs for tuple-based data provenance. Furthermore, we develop a prototype combining streaming and manually sampled data to realize our approach.

This paper is structured as follows. In Section 2, we discuss related work. Next, we provide a detailed description of our motivating scenario followed by the problem description in Section 4. In Section 5, we provide the structure of our temporal data model followed by the implementation

VLDB '10, September 13-17, 2010, Singapore

that demonstrates the viability of our approach in Section 6. Finally, we conclude with the hints of some future work.

2. RELATED WORK

Recently issues pertaining to data provenance are getting more attention from researchers. In [3], authors have described a data model to compute provenance on both relations and tuples level. In this data model, the location of any piece of data can be uniquely described by a path. This paper shows case studies for traditional data but it does not address how to handle streaming data and associated overlapping windows. In [10], authors have provided a data model for provenance repository which is based on relational database. Their approach maintains relation-based data provenance whereas our approach provides fine grained data provenance.

Relation-based data provenance cannot reproduce results. The work described in [12] proposed a data and collection model using timestamp based approach to collect provenance information when there is any change in sampling rate and accuracy of the stream. Though it saves a lot of disk space, it cannot address update of sampled data. We use multiple timestamps to identify the validity of a particular (updated) tuple. In [5] authors have presented an algorithm for lineage tracing in a data warehouse environment. They have provided data provenance on tuple level. Their algorithm only works for traditional data. It cannot address the issue of database state change due to update.

For databases that change over time, compact versioning is essential to recover data referenced by the provenance of data derived from an earlier version of the database [2]. Since one version is an extension of the previous version, this versioning technique incurs space overhead. Our proposed approach does not store any versions physically in the disk instead we attach timestamps to each tuple to retrieve any database state based on a given timestamp.

In e-science applications, supporting reproducibility of research results are necessary. In [8], authors outline the structure of a provenance-aware storage where provenance data will be treated as the first class data. For recording and querying provenance data Tupelo2 project [7] has been initiated. This project is aimed at creating a metadata management system which stores annotation triples (subject-predicate-object) in several kinds of databases, including normal relational databases. Tupelo2 cannot address issue with update operation.

Recently, a complete DBMS, LIVE [9] can store base and derived relations with simple versioning capabilities where each tuple includes a start and end version number. LIVE also preserves the lineage of derived data items. Since LIVE uses different version number associated with each relation, we cannot retrieve the overall database state given a single version number. Our approach of using timestamp instead of version number overcomes this drawback. Most significantly, in our approach, we need not maintain any tuple-based provenance data instead we store relation-based provenance data along with temporal data model which is more cost effective than LIVE.

3. SCENARIO

Figure 1 depicts a Bluetooth localization scenario that has been set up in our SensorDataLab [17]. The location of a user is determined by acquiring the signal strength of a Bluetooth device carried by the user and the known location of the system acquiring the signal strength measurement. Linksys NSLU2 devices are used for acquiring signal strength measurements of all Bluetooth devices. On these systems, a Bluetooth discovery application is installed which continuously checks for handheld devices and reports detected devices via a UDP packet to the data processing system. A packet contains the person's device MAC address, identification number of the discovery systems, signal strength, and the timestamp (see figure 6). The NSLU2 systems are represented as EWI 1148, EWI 1149, EWI 1150.

In addition, the deployment location of NSLU2 device as well as the mapping of MAC address of a handheld device to an actual person at a specific point in time is documented and made available as manually sampled data. The data processing allows to correlate streaming and sampled data and provides a query interface to access the data on-line and off-line.

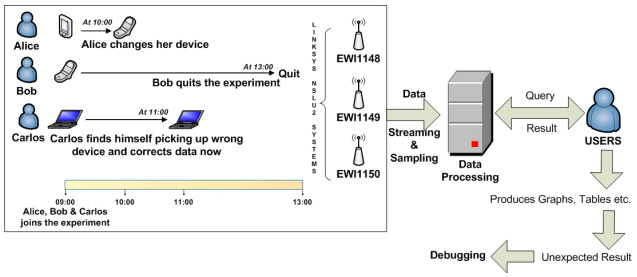


Figure 1: Bluetooth localization scenario in SensorDataLab

In our scenario Alice, Bob, and Carlos are three users, joining the experiment on 2010-03-03 at 9:00. Each of them using a mobile device. It turns out that the mobile device of Alice has not been charged over night and is running out of power. Therefore, at 10:00 she has to exchange the mobile device with another one. Bob has to attend a lecture in the afternoon and therefore is leaving the experiment at 13:00. At around 11:00 Carlos finds out that he picked up a new, unused mobile device instead which had been assigned to him. Since Carlos likes this mobile device better, the device had been permanently assigned to Carlos.

The supervisor of Alice, Bob and Carlos uses the data for publishing a paper about a new localization approach tested in this experiment. From the available data, she evaluates her approach and creates graphs to document the results. If the outcome is unexpected, she may want to debug the results. Thus, this scenario exhibits the properties of an e-science scenario, since she must be able to reproduce the evaluation results and graphs later. The requirement of reproducible results corresponds to tuple-based provenance in the scenario as described above because it documents how each tuple has been created. For the rest of this paper, we will explain our approach to achieve fine grained data provenance based on this scenario.

4. PROBLEM DESCRIPTION

4.1 Fine grained Data Provenance

In our scenario (see Section 3), each second a lot of streaming data is arriving from different sensor nodes. Moreover

manually sampled data including metadata are also stored to help the overall data processing job. We have 8 NSLU2 devices installed in our lab. Whenever users are roaming around the lab, each second a UDP packet is sent containing the user's device MAC address, detection timestamp along with other parameters.

Now, consider a time-triggered query to compute a person's location based on the readings of the last 30 seconds. This translates in a continuous query having window size of 30 seconds. At each second, 8 different tuples/packets will be sent by those NSLU2 devices. After each second, the window shifts forward by a second. Therefore, at a particular moment we have 240 data tuples which should be processed to compute that time-triggered query. For each data tuple, we associate provenance data using a pointer to the tuple represented as *bigint* field. Moreover, we need two more pointers to point to the activity and the resulting output tuple assuming there is only one output tuple. In total, we need to preserve 242 pointers in order to have the tuple-based data provenance. In MySQL[15], the *bigint* field consumes 8 bytes. The output will be current location of a particular person which is nothing but a co-ordinate in form of (x,y) and consumes 8 bytes in total. Therefore, the ratio of the provenance data to processed data is 242:1 per processed data tuple in this scenario. In other words, only 4 gigabytes of a 1 terabyte disk will be used to store the sensor data and the rest of the space will be consumed by provenance data. Moreover, provenance data is a type of indirection used to identify the original data which has no significant meaning to users. Therefore, buying additional storage space seems to be an expensive solution to this problem.

Based on the example described above in this section, relation-based data provenance needs to preserve only three provenance data. One for the set of input data, another for the query and the rest is maintained for the output. For relation-based data provenance, the ratio of provenance data to actual desired sensor data is 3:1 per processed query and it is independent of overlapping window size between two subsequent windows and number of tuples per second. Therefore, from the storage point of view relation-based data provenance is more efficient than tuple-based data provenance.

4.2 Reproducible Results

Reproducibility of results can be achieved by having different versions of a database - a new version after every change of the database. Traditionally, versioning is implemented by replicating the complete database before applying the change. An alternative way is to document changes in the database according to time. Timestamps can be used as a global version number. Using timestamp, we can provide a particular state of the database without storing versions physically. In this paper, we achieve reproducibility by using timestamps as version numbers and requiring a *consistency* property on the database which ensures for a query on a particular database state in the past to have the same result set regardless of the query execution time. The definition of *consistency* is given below.

Definition 1. If a particular query is executed on the same database state by same/different users at different points in time, users are expecting to have the same result sets each time under the assumption that the query processing is not hindered by any means of network volatility.

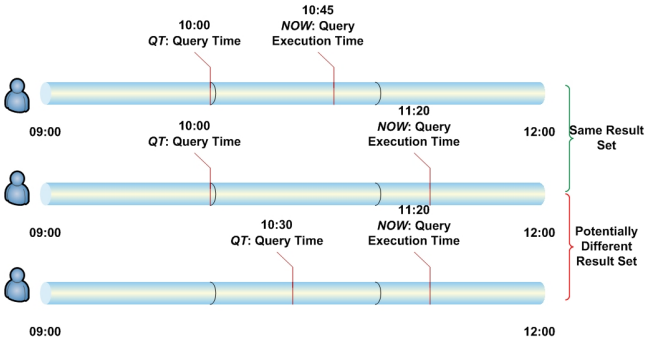


Figure 2: Query Time and Query Execution Time

In the definition, the term *same result set* refers to the same set of tuples extracted from the same set of relations from participating nodes for the same query executed at different points in time. Figure 2 pictorially represents definition 1. Assume that a user wants to know the location of Alice on a particular point in time which can be termed as *query time, QT* is represented as query Q at QT. In the upper pair of timelines, a user submits the query Q at QT=10:00. Regardless of their query execution time which is represented as *NOW*, the outcome should be the same since they queried on the same database state that is available on 10:00. On the other hand, QT is different for the lower two timelines. The middle timeline shows that the user submits query Q at QT=10:00 and the last timeline depicts that the user submits Q at QT=10:30. Though these two queries are executed at the same point in time, the result set may be potentially different if there is a change on Alice's handheld device. In other words, if there is a database state change we may get potentially different results for a particular query. The consistency property is also applicable for continuous queries. Reconstructing the window having same set of tuples and trigger condition will ideally produce the same result set each time irrespective of the *query execution time*. This is how definition 1 differentiates between *query time* and *query execution time* which allows users to have the same set of data retrieved as a result of the query depending on the point in time for which they want the query result, but irrespective of the time when the query has been executed. It fulfills the requirement of retrieving historic data as well as provides a consistent view of the database.

4.3 Data Classes

In our scenario, we have both streaming and manually sampled data. Streaming data is automatically acquired from sensors which is not the case for manually sampled data. The volume of streaming data is much larger than the volume of sampling data. Manually sampled data or metadata are associated with streaming data which make them important to preserve into the database. Sometimes, there is a large time delay between a fact becomes valid in the real world and that fact is inserted into the database. This delay is known as *propagation delay*. Since human intervention is needed to enter sampled data in database, it may have longer propagation delay than streaming data. Moreover, data processing is also challenged by update of

sampling data. Streaming data, on the other hand, never updates but due to its high volume, it is a challenging task to provide tuple-based data provenance in streaming scenarios. Next sections will discuss these problems associated with streaming and sampling data in detail.

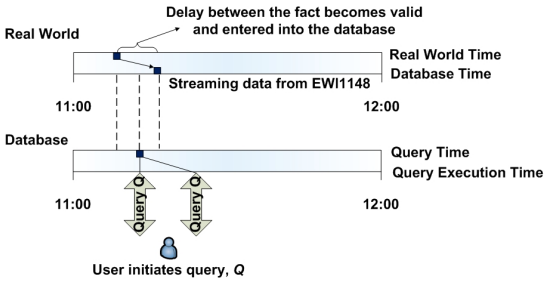


Figure 3: Propagation delay of streaming data

4.4 Propagation Delay in Streaming Data

In Figure 3, we have one continuous timeline and two different worlds: real world and database world. The upper block of timeline shows that the data tuples generated from EWI 1148 has delay between the time they got valid and entered into the database due to the propagation delay. This delay can affect the data processing steps and eventually the outcome. In the lower block of the processing timeline, we see a user initiates a query Q with same *query time* on tuples generated by EWI 1148 in two different points of *query execution time*. When the query Q is executed for the first time, as tuples are yet to enter in the database the outcome contains no result which is unexpected. The set of data arrives later after the first query execution and influence the outcome of the next execution of the same query Q. Maintaining relation-based provenance in the aforementioned scenario, cannot extract original data to reproduce results.

4.5 Updates in Sampled Data

Sampling data may be updated or modified over time. Figure 4 shows an example where user Alice changes her handheld device at 10:00 (see section3). After changing the device, the related data on Alice's new handheld device (e.g. device MAC) is entered into the database and overwrites the previous data. This operation indicates a state change for the overall database. Now, if a query Q on Alice's handheld device is executed on two different points in time (different query execution times), we will get two different results because of the state change of the database. Therefore, this update operation in the database causes to have inconsistent results and thus creates inconsistency in the database.

5. TEMPORAL DATA MODEL

One of the major challenges to preserve our consistency property 1 is to allow query execution on the same database state. That's why, we use a temporal data model to avoid storing of all the previous versions physically. Using a temporal model, we can retrieve any particular state of the database based on a given timestamp since each data tuple will be associated with temporal attributes. Our proposed data model is actually inspired by the bi-temporal data model [11] using the following temporal attributes:

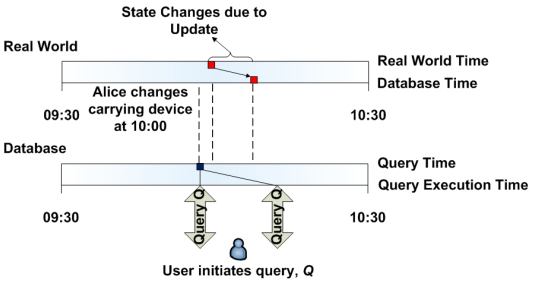


Figure 4: Update of sampled data

- **valid time** representing the point in time a sample has been taken or a measurement has been sensed.
- **transaction time from** is the point in time the tuple has been inserted in the database.
- **transaction time to** representing the point in time the tuple is marked as deleted without physically deleting it.

These temporal attributes allow users to initiate queries on a database mentioning a specific timestamp. Next, we discuss the way of executing some most common database operations based on our data model. Since streaming data never changes, only *insert* operation is applicable for streaming data.

- **Insert**: A tuple is added in the database for the very first time with specified *valid time* and *transaction time from* being the current point in time (e.g. Alice, Bob and Carlos join the experiment). The value of *transaction time to* is set to '0:00:00'.
- **Update**: It addresses the situation whenever a user would like to rectify wrong data given earlier (e.g. Carlos uses one device but another device was registered for him). *Transaction time to* of the existing tuple is set to $NOW - 1$ and a new tuple is added to the database with same *valid time* as the existing tuple. *Transaction time from* is set to NOW and *transaction time to* is set to '0:00:00'. The difference from *change of data* operation is that here the *valid time* of existing and new tuples are same.
- **Delete**: It refers to the incident that causes damage or complete removal of a particular entity from the scenario (e.g. at 13:00 Bob is not participating in the experiment anymore). In the tuple describing the participation of Bob in the experiment is updated by setting the value of *transaction time to* to NOW-1.

One of the principle requirements of our data model is to preserve all the past data in order to execute queries on a given database state so that we can maintain consistency according to the given definition 1. We are not going to delete or modify any existing tuples in the database rather we will insert new tuples with different *valid* and *transaction times*. Therefore, all these database operations need to be handled in a different manner than a traditional database does.

6. IMPLEMENTATION

6.1 Prototype

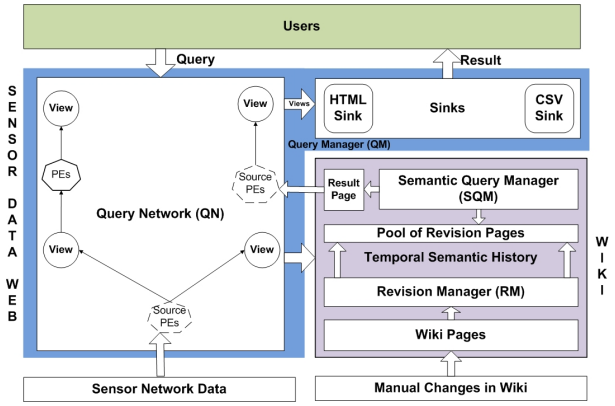


Figure 5: Architecture of prototype

We build a prototype to validate our approach of achieving fine grained data provenance for both streaming and sampled data which ensures to reproduce query results. We make use of Sensor Data Web [18] for gathering, processing and publishing sensor data. We perform some modifications on the existing java code so that we can realize and execute our proposed approach.

Figure 5 shows the basic building block of the platform. In sensor data web platform, *Query Manager* (QM) is responsible for collecting streaming data from sources like GSN [13] and sampling data from the wiki via a Sparql end point. Within the query manager a query network is generated, consisting of several processing elements (PEs). Some of them are source PEs which can communicate and receive streaming data directly from nodes in the sensor network or pull information from external sources. Every PE presents output as a view. A view is not considered as the final outcome since it can be input for another PE. We modify the structure of original views to ensure that the *transaction time* for each data tuple is now included into the views. Users can request results in a preferred format like as a HTML page or a CSV document. This request is handled by *sinks* which return results to users in requested format via specific sink (e.g. HTML sink, CSV sink). We add one extra parameter *query time* in each sink structure so that each of these sinks now can return results based on the given *timestamp*.

Sensor data web can also interact with external sources to pull sampled data according to a given query. In order to manage sampled data, we use MediaWiki [14] as our basic platform. One of the main reasons for choosing MediaWiki is to collaborate with different metadata and sampled data owners. As wiki is well known for it's community based use, using wiki as the repository of sampling data would be an easy way to collect those data. On the top of MediaWiki, we use semantic mediawiki extension [16] on top of which we build our own semantic wiki extension known as Temporal Semantic History [19].

Our developed extension tracks and monitors the content of each page in wiki. Data are changed manually in a wiki page. Revision manager (RM) preserves the previous content after each revision done on a particular page according to *timestamp* in a new *revision page*. Each revision page keeps the value of (e.g. *valid time, transaction time from*

timestamp	system_id	mac_address	rssi	valid time	transaction time
1267610610	ewi1148	00:15:F2:B6:C4:D5	75	2010-03-03 11:03:30	2010-03-03 11:03:36
1267610638	ewi1148	00:15:F2:B6:C4:D5	77	2010-03-03 11:03:58	2010-03-03 11:04:06
1267610669	ewi1148	00:1E:45:D8:C0:D1	78	2010-03-03 11:04:29	2010-03-03 11:04:36
1267610696	ewi1148	00:1E:45:D8:C0:D1	81	2010-03-03 11:04:56	2010-03-03 11:05:06
1267610731	ewi1148	00:15:F2:B6:C4:D5	73	2010-03-03 11:05:31	2010-03-03 11:05:35

Figure 6: A set of streaming data

and *transaction time to*) along with other data. The value for *transaction time from* and *transaction time to* are added into the wiki page by the system itself. These revision pages together form the pool of revision pages. When a query Q requests data from wiki, it is redirected via *query network* to this extension. *Semantic query manager* (SQM) chooses appropriate revision pages from the pool according to the user given timestamp in Q. Then the content of selected revision page is transferred and displayed in the result page. This data is provided as input to one of the source PEs which can handle sparql data. Then the data is further processed and result is sent to users.

6.2 Use Case

In the proposed data model, each data tuple is associated with temporal attributes irrespective of their sources and types. Figure 6 shows a set of streaming data produced by one of the NSLU2 systems, EWI 1148 in our scenario. The temporal attributes *valid time* and *transaction time* (as an abbreviation of *transaction time from*) are added for each tuple. *Transaction time to* is not needed for streaming data, since we consider *append-only* streaming data.

On the other hand, sampled data (e.g. manually sampled data, metadata) is stored in a semantic wiki. In a wiki, data is stored in form of SPO (Subject-Predicate-Object) triples. Figure 7 depicts that for each entity, a unique wiki page is created based on the *candidate key*. As for example, we have three different persons in our scenario: Alice, Bob and Carlos and a unique wiki page is created according to the person name. In triple store, for all triples, *subject* contains name of the page. Moreover, once a wiki page is modified, the page content before the revision is preserved in order to provide original data upon user requests. The name of these revision pages depends on the original page name and *transaction time from*. Moreover, '0:00:00' in *transaction time to* attribute is used as a pattern to indicate that the tuple is currently valid.

Sampling data may be updated and deleted over time. As discussed earlier, sampling data is organized in a semantic wiki which has different data organization technique. Among the different database operations, *update* is a more interesting operation for sampling data. In figure 7, Alice changed her device after a while which is indicated by tuple no. 4. As overwrite existing data causes problems to retrieve original data, we insert another tuple having different *valid* and *transaction time from*. Before inserting the new tuple, we update *transaction time to* of the previous tuple to NOW-1.

6.3 Discussion

Our approach achieves fine grained data provenance with reduced storage costs. Consider the set of streaming data in figure 6. If we perform any *select*, *project* or *join* operations on that dataset, we will get output tuples. Now,

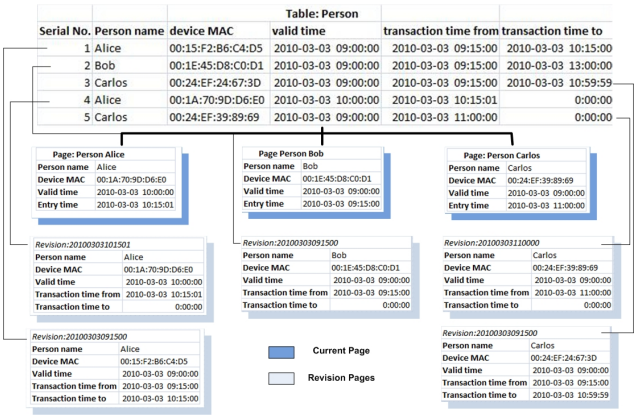


Figure 7: Organization of Sampled data in Wiki

based on a user given timestamp, we can retrieve original database state at that point in time. Then, we will use coarse grained provenance data to figure out the tuples from input dataset which participated in the query to produce output data tuples. This is how, we can achieve fine grained data provenance with reduced storage costs by maintaining coarse grained data provenance and applying temporal data model.

In section 4.1, a comparison of consumption of storage space between fine grained and coarse grained provenance data has been given. In our prototype, for each tuple, we need at most three timestamp attributes which take at most 12 bytes storage space per tuple and it is independent of window size, size of the overlap of the windows, and number of tuples per second. In tuple-based provenance, each data tuple is associated with provenance data which is a pointer to the tuple itself and since a particular data tuple is participating in the query execution for several times depending on the overlap of subsequent sliding windows, the space consumed for provenance data is much larger than our proposed approach. If there is no overlap between subsequent sliding windows, our approach incurs extra disk space (8 bytes per tuple) as much as fine grained provenance does. Though our prototype requires more space than relation-based data provenance, it enables users to have reproducible results and overcomes drawbacks of relation-based data provenance.

We assume the *append-only* data stream processing engine in our scenario. There are some stream processing engines which act as *non append-only*. In those cases, our solution will handle stream data in a similar way of handling manually sampled data.

7. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an approach to achieve fine grained data provenance with low storage costs. To achieve our goal, we maintained relation-based data provenance along with timestamp-based logical versioning of the database. The proposed approach is mainly beneficial for streaming data, thus data processed on-line. However, the approach allows us also to combine and correlate streaming and sampled data. The proposed approach has been implemented for both of the streaming and sampled data which shows the viability of our approach.

In future, we would like to compare performance (i.e. storage cost, response time) of our approach to any existing techniques.

8. REFERENCES

[1] D. Brus and M. Knotters. Sampling design for compliance monitoring of surface water quality: A case study in a polder area. *Water Resources Research*, 44(11):95 – 102, 2008.

[2] P. Buneman, S. Khanna, and T. Wang-Chiew. Data provenance: Some basic issues. *Foundations of Software Technology and Theoretical Computer Science*, pages 87–93, 2000.

[3] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and where: A characterization of data provenance. *Database Theory - ICDT 2001*, pages 316–330.

[4] P. Buneman and T. Wang-Chiew. Provenance in databases. In *Proc. Intl. Conf. on Management of data*, pages 1171–1173, New York, NY, USA, 2007. ACM.

[5] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, vol. 12, pages 41–58.

[6] J. de Gruijter, D. Brus, M. Bierkens, and M. Knotters. *Sampling for natural resource monitoring*. Springer Verlag, 2006.

[7] J. Futrelle. Tupelo Server. Website. `http://tupeloproject.ncsa.uiuc.edu/`.

[8] J. Ledlie, C. Ng, D. A. Holland, K. kumar Muniswamy-reddy, U. Braun, and M. Seltzer. Provenance-aware sensor data storage. In *Workshop on Networking Meets Databases (NetDB)*, 2005.

[9] A. Sarma, M. Theobald, and J. Widom. LIVE: A Lineage-Supported Versioned DBMS. In *Proc. Intl. Conf. on Scientific and Statistical Database Management*, 2010.

[10] M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 603–620.

[11] C. K. University and C. Koncilia. A bi-temporal data warehouse model. In *Proc. Intl. Conf. on Advanced Information Systems Engineering*, pages 77–80, 2003.

[12] N. N. Vijayakumar and B. Plale. Towards low overhead provenance tracking in near real-time stream filtering. In *Provenance and Annotation of Data*, pages 46–54, 2006.

[13] Website. Global Sensor Network. `http://www.swiss-experiment.ch/index.php/GSN:Home`.

[14] Website. Mediawiki. `http://www.mediawiki.org/wiki/MediaWiki`.

[15] Website. MySQL. `http://www.mysql.com/`.

[16] Website. Semantic Mediawiki Extension. `http://www.mediawiki.org/wiki/Extension:Semantic_MediaWiki`.

[17] Website. Sensor Data Lab. `http://www.sensordatalab.org/wiki/index.php5/Loc:Home`.

[18] Website. Sensor Data Web. `https://sourceforge.net/projects/sensordataweb/`.

[19] Website. Temporal Semantic History. `http://www.sensordatalab.org/wiki/index.php5/Extensions:Temporal_Semantic_History`.

Processing Nested Complex Sequence Pattern Queries over Event Streams

Mo Liu, Medhabi Ray, Elke A. Rundensteiner, Daniel J. Dougherty
Worcester Polytechnic Institute, Worcester, MA 01609, USA
(liumo|medhabi|rundenst|dd)@cs.wpi.edu
Chetan Gupta, Song Wang, Ismail Ari‡, Abhay Mehta
USA Hewlett-Packard Labs, USA
‡Ozyegin University, Turkey
(chetan.gupta|songw|abhay.mehta)@hp.com ‡Ismail.Ari@ozyegin.edu.tr

ABSTRACT

Complex event processing (CEP) has become increasingly important for tracking and monitoring applications ranging from health care, supply chain management to surveillance. These monitoring applications submit complex event queries to track sequences of events that match a given pattern. As these systems mature the need for increasingly complex nested sequence queries arises, while the state-of-the-art CEP systems mostly focus on the execution of flat sequence queries only. In this paper, we now introduce an iterative execution strategy for nested CEP queries composed of sequence, negation, AND and OR operators. Lastly the promise of applying selective caching of intermediate results to optimize the execution. Our experimental study using real-world stock trades evaluates the performance of our proposed iterative execution strategy for different query types.

1. INTRODUCTION

Complex event processing (CEP) has become increasingly important in modern applications, ranging from supply chain management for RFID tracking to real-time intrusion detection [1, 2, 3]. CEP must be able to support sophisticated pattern matching on real time event streams including the arbitrary nesting of sequence operators and the flexible use of negation in such nested sequences. For example, consider reporting contaminated medical equipments in a hospital [4, 5]. Let us assume that the tools for medical operations are RFID-tagged. The system monitors the histories of the equipment (such as, records of surgical usage, of washing, sharpening and disinfection). When a healthcare worker puts a box of surgical tools into a surgical table equipped with RFID readers, the computer would display approximate warnings such as "This tool must be disposed". A query Q_1 = *SEQ (Recycle r, Washing w, NOT SEQ(Sharpening s, Disinfection d, Checking c), Operating op) with the condition that ([ID] (equality on ID) and op.ins-type = "surgery")* expresses this critical condition that after being recycled and washed, a surgery tool is being put back into use without first being sharpened, disinfected and then checked for quality assurance. Such complex sequence queries contain complex negation specifying the non-occurrence of composite event instances, such as negating the composite event of sharpened, disinfected and checked subsequences.

However, the state-of-the-art CEP in the literature including SASE [1] and ZStream [3] do not support such nested queries. Even though the Cayuga system [2] mentions composable queries, they assume the negation filter is only applied to a single primitive event type within the SEQ pattern. Our objective however is to allow the specification of negation within any level of the nested query as in the above example. While CEDR [6] allows applying negation over composite event types within their proposed language, the execution strategy for such nested queries is not discussed. In short, no processing mechanisms for nested complex negation of CEP queries have been discussed in the literature to date. In this work, we address this gap by designing an execution strategy specifically to handle nested CEP queries specified by the nested complex expression query language NEEL[1]. The semantics of this language is presented in [7].

Our contributions in this paper include:

- We introduce an algebraic query plan for nested CEP queries expressed in NEEL.
- We design an iterative topdown execution strategy based on the algebraic plan that applies a window constraint tightening technique designed to correctly process nested sub-queries. Intermediate results are pushed up conservatively for delayed resolution when a child query can't be fully answered locally for nested negation.
- We experimentally evaluate our proposed execution strategy studying nested queries with different properties including sub-query lengths and nesting levels on real data streams.
- Lastly selective caching of intermediate results is introduced as technique for optimizing the execution.

2. NESTED CEP QUERY MODEL

2.1 Event Model

An event instance is an occurrence of interest which can be either primitive or composite as further introduced below. *A primitive event instance* denoted by a lower-case letter (e.g.,'e') is the smallest, atomic occurrence of interest in a system. e_i.ts and e_i.te denote the start and the end timestamp of an event instance e_i, respectively, with e_i.ts $\leq e_i$.te. For a primitive event instance e, e_i.ts = e_i.te. For simplicity, we use the subscript i attached to a primitive instance e to denote the timestamp *i*.

[1]NEEL stands for **Ne**sted Complex **E**vent Query **L**anguage.

A composite event instance is composed of constituent primitive event instances $e = < e_1, e_2, ..., e_n >$. A composite event instance e occurs over an interval. The start and end timestamps of e are equal to $\min\{e_i.\text{ts} \mid e_i \text{ in e}\}$ and $\max\{e_i.\text{te} \mid e_i \text{ in e}\}$, respectively.

An *event type* is denoted by a capital letter, say E_i. An event type E_i describes a set of attributes that the event instances of this type share. An event type can be either a primitive or a composite event type [8]. *Primitive event types* are pre-defined in the application domain of interest. *Composite event types* are aggregated event types created by combining other primitive and/or composite event types. $e_i \in E_j$ denotes that e_i is an instance of the type E_j. Suppose one of the attributes of E_j is attr and $e_i \in E_j$, then we use e_i.attr to denote e_i's value for that attribute.

2.2 The Nested Complex Pattern Query Language NEEL

We now briefly introduce the *NEEL* query language for specifying complex nested event pattern queries [1, 6, 9] as an extension of basic non-nested languages from the literature. *NEEL* supports the nesting of AND, OR, Negation and SEQ operators at any query nesting level as in Table 1.

<**Query**>::= PATTERN <event-expression> WITHIN <window> [RETURN <output-specification>]
<event-expression> = <ex>
<ex> ::= **SEQ**((<ex> | **!** (<ex>, [<q>]))*,<ex>, (<ex> | **!** (<ex>, [<q>]))*, [<q>]) | **AND**((<ex>, (<ex> | **!** (<ex>, [<q>]))*, [<q>]) | **OR**((<ex>)+, [<q>]) | (<primitive-event type>, [<var>])
<primitive-event type> ::= E_1 | E_2 | ... <var> ::= event variable e_i <q>::= (<elemqual>)* <elemqual> ::= <var>.attr <op> <var>.attr | <var>.attr <op> constant <op> ::= < | > | ≤ | ≥ | = | != <window>::= time duration w | tuple count c

Table 1: NEEL Query Language

A primitive event type E_i itself is an event expression. If E_1, E_2 ,..., E_n are event expressions, an application of SEQ, AND and OR over these event expressions is again an event expression [8]. In other words, nesting of AND, OR and SEQ operators is supported.

SEQ in the PATTERN clause specifies a particular order in which the event instances of interest should occur. If there is a ! (NOT) symbol before an event expression in an operator, we say that the event expression marked by ! is to be negated. Event instances that satisfy the positive components with no events in the stream relative to this match satisfying the negative components are output. If several adjacent event types are marked by ! in a SEQ operator such as SEQ(E_1, ! E_2, ! E_3, E_4), the query requires the non-existence of any E_2 and E_3 events in either order between E_1 and E_4 events within the input stream. In other words $< e_1, e_3, e_4 >$ and $< e_1, e_2, e_4 >$, $< e_1, e_3, e_2, e_4 >$ and $< e_1, e_2, e_3, e_4 >$ all do not result in a valid match for this query.

An event expression exp_i can be used as a component in SEQ, AND and OR operators to construct another expression exp_j. Then we call exp_j the *outer* or parent *expression* of exp_i and exp_i the *inner* (or child) *expression* of exp_j. Qualification in the PATTERN clause contains predicates on single attributes or on attributes across multiple event types in the query [6, 1]. The event variables defined in an outer expression are visible within the scope of its own nested inner expressions. Local predicates are specified directly inside exp_i. Correlated predicates involving events from both an outer and an inner expression are associated with the innermost expression that define an event in the predicate. Correlated predicates involving two adjacent sibling expressions are not allowed since the events in one inner expression are not visible in any sibling.

The WITHIN clause indicates the temporal interval within which the event instances of interest must occur. The RETURN clause transforms the set of matching event instances extracted by the query into a complex event as specified in the output specification.

Q_1 below in Figure 1 is a sample query expressed by *NEEL*.

```
PATTERN  SEQ(Recycle r, Washing w,
              ! SEQ(Sharpening s, Disinfection d, Checking c, s.id=d.id=c.id=o.id),
                Operating o, r.id=w.id=o.id and o.ins-type="surgery")
WITHIN   1 hour
```

Figure 1: Sample Query Q_1 for Hospital Hygiene

2.3 Nested CEP Query Plan

A query expressed by a *NEEL* specification is translated into a default algebraic query plan composed of the following algebraic operators: Window Sequence (*WinSeq*), Window Or (*WinOr*) and Window And (*WinAnd*). During query transformation, each expression in the event pattern is mapped to one operator node in the query plan. The same window w is assigned to all operator nodes. *WinSeq* first extracts all matches to the positive components specified in the query, and then filters out events based on negative components as specified in the query. *WinOr* returns an event e if e matches any one of the event expressions specified in the WinOr operator. *WinAnd* computes the cross product of its positive components. For queries expressed by *NEEL*, predicates are placed into the respective algebra operators in the nested event expressions (see Section 2.2).

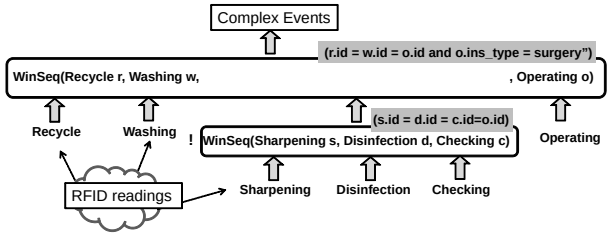


Figure 2: Basic Query Plan

EXAMPLE 1. *Figure 2 depicts the query plan for query Q_1 in Figure 1. The two SEQ expressions in Q_1 are transformed into two WinSeq operator nodes in the plan. The predicate s.id = d.id = c.id = o.id is placed with the inner WinSeq operator node containing the negative component. The other predicates are attached to the topmost WinSeq operator node.*

3. NESTED CEP QUERY PROCESSING

3.1 Execution of Individual Operators

For simplicity, we briefly review the implementation strategy of one of the operators, namely, the SEQ operator, while the others can be implemented in a similar fashion. We adopt the state-of-the-art stack-based strategy for SEQ execution [1, 10, 11]. We associate a stack with each event type in the query. Each received event instance is simply appended to the end of the stack of its type. Event instances are augmented with pointers ptr_i to adjacent events to facilitate quick locating of related events in other stacks during result construction.

The arrival of an event instance e_m of the last event type E_m of a query q_i triggers the compute function of q_i[2]. The result construction is done by a depth first search along instance pointers ptr_i rooted at that last arrived instance e_m. All paths composed of edges "reachable" by that root e_m correspond to one matching event sequence returned for q_i. When negative event types are specified in WinSeq, then during sequence construction any edges "reachable" from the root e_m are skipped if an instance of the negative event type is found in the corresponding stream position. Events that are outdated based on the window constraints are purged.

3.2 Iterative Nested Execution Strategy

Following the principle of nested query execution for SQL queries [12, 13, 14, 15], the outer query is evaluated first followed by its inner sub-queries. The results of the inner queries are passed up and joined with the results of the outer query. The main idea of our nested execution is about passing down more stringent window constraints from outer queries to inner queries. For every outer partial query result, a constraint window (see Figure 3) is passed down for processing each of its children sub-queries. These sub-queries compute results involving events within the substream constrained by the constraint window. Qualified result sequences of the inner operators are passed up to the parent operator and the outer operator then joins its own local results with that of its positive sub-queries. The outer sequence result is filtered if the result set of any of its negative sub-queries is not empty. We apply iterative execution until a final result sequence is produced by the root operator. Finally, the process repeats when the outer query consumes the next instance e. We will discuss nested queries with negation and predicates in more detail in Sections 3.3 and 3.4, respectively.

```
Interval IntervalConstraints (Result r_j, Query q_i)
// r_j is one partial result of the outer query
01 Interval ts;
02 if(root operator of q_i is SEQ)
   // gets the position of q_i in outer query
03 { nestedPosition = getNestedPos(q_i);
       // if outer query starts with sub query q_i
04 if(nestedPosition == 0)
       // left bound is time of last event in result r_j - W
05     ts_left = getTime(r_j.LastEve) - W;
       // if outer query ends with sub query q_i
06 if(nestedPosition == r_j.size)
       // right bound is time of first event in result r_j + W
07     ts_right = getTime(r_j.FirstEve) + W;
08 else
09     {ts_left=getTime(r_j.get(nestedPos-1))
10     ts_right=getTime(r_j.get(nestedPos))}
11 if(root operator of q_i is AND)
12 {ts_left = getTime(r_j.lastEve) - W;
13 ts_right = getTime(r_j.lastEve); }
14 if(root operator of q_i is OR)
15 {ts_left = getTime(r_j.lastEve) - W;
16 ts_right = getTime(r_j.lastEve); }
17 return ts;
```

Figure 3: Algorithm to Compute Interval Constraints for an Inner Query Q_i Given an Outer Partial Result r_j

3.3 Processing Nested Queries with Negation

We now describe our approach of supporting negations in nested queries. In SASE [1, 11, 10], flat queries can have negations and they are dealt with using the timestamp information. More precisely, if a query has a negative A between positive B and C event types, they first evaluate the query without the negation, i.e., they compute all B-C pairs. Then for every result generated they check if an A event occurred between the qualified B and C events. If it occurs, such pairs are discarded. When two negative event types are adjacent to each other, their order does not matter. For example, SEQ(A, !B, !C, D) is equivalent to SEQ(A, !C, !B, D). That is, all (A, D) result pairs without any B and C events in between them would be returned.

For negative event types at the end of a query, postponed sequence evaluation is applied. That is the execution is continued till the last negation as per our iterative strategy however results are not output. Instead at t he arrival of every new event we note the time stamp of the event and also check whether it is a triggering event for the last negative part of the query. If it is not a triggering event, based on the time stamp of the arriving event, some results from the buffer may be output and removed from the buffer. If it is a triggering event, the negative part of the query is executed and if it produces some partial results, the result buffers of the outer query are completely cleared. However if the negative ending part of the query does not produce any results, some results are output and removed from the result buffers based on the time stamp of the arriving event.

In our nested query model, a sub-query as a whole could also be negated. For example, SEQ(A, ! AND(B, C), D). For each outer result of SEQ(A, D), we search for AND(B, C) results occurring between such A and D events. If none exist, then the outer SEQ(A, D) result is returned, otherwise it is filtered out.

We distinguish between the following positions in which the negation clause can occur.

- *Bound by Upper Query.* The existence of a negative event instance could be bounded by positive event instances in the direct upper queries. Examples of this category include SEQ(A, !B, C) and SEQ(A, SEQ(B, !C), D). In the second query, negative C events are bound by B and D events. B events that do not have any C events occurring after them and before D events are passed up to the upper query operator. All B events passed up will be joined with the outer SEQ(A, D) result to construct SEQ(A, SEQ(B, !C), D) results.

- *Bound by Adjacent Query.* The existence of a negative event instance could be bound by positive event instances of an adjacent sibling sub-query. Examples of this type include SEQ(A, SEQ(B, !C), SEQ(D, E), F) or SEQ(A, !B, SEQ(C, D), E). In this case, we apply a contextual delayed constraint technique. Namely, we conservatively pass up additional intermediate results as compared to the case described above. In SEQ(A, SEQ(B, !C), SEQ(D, E), F), outer SEQ(A, F) results $< a_i, f_j >$ are constructed. The constraint window for both children sub-queries SEQ(B, !C) and SEQ(D, E) is [a_i.te, f_j.ts]. When processing the sub-query SEQ(B, !C) within this constraint window, any event of type B should be passed up. We cannot filter out events of type B even though C events exist after it within its constraint window. The reason is that the right bound of the interval constraint of the query SEQ(B, !C) is decided by the results of the query SEQ(D, E). We should not have a C event between a B or D event. However, it is not possible to know time stamps of D events while still processing the query SEQ(B, !C). Hence the decision is postponed until the results of both the inner queries are returned to the outer query and then the filtering of results takes place based on the presence of C events.

3.4 Processing Nested Queries with Predicates

The approach of handling sub-queries with correlated predicates is similar to the basic nested execution described above except that the join is not only based on timestamps but also on other predicates. Below, we list the different cases for predicate handling.

[2]if E_m is a negative event type, postponed sequence evaluation is applied. We omit the details here.

- *Local predicates.* Events are filtered based on predicate values before being stored in their stack. Query processing proceeds otherwise as explained above. For example, for the query in Figure 2, Operating events where the instrument type is not equal to "surgery" will be filtered.

- *Correlated predicates between inner and outer queries.* Nested sub-queries may be correlated with their parent queries by means of predicates. In order to evaluate these queries with predicates, it is necessary to pass down attribute values to the children queries. For example, the query in Figure 2 requires events in the inner sub-queries have the same tool id as the outer match. For each outer SEQ(Recycle r, Disinfection d, Operating o) match, the tool id information for the operating instance is thus passed down to the children sub-queries. Inner query results involving events having the same tool id with the outer match are returned to the upper query. As can be seen in Table 1, predicates on negative components are associated directly with the later and not with the operator as a whole. They are thus only evaluated for those subqueries, for which the positive parent context match has already been established.

3.5 Putting It All Together

At compile time, queries with negation bounded by an adjacent sub-query (as discussed in Section 3.3) are marked with label "delayed constraint". More specifically, if a query q_i is labeled as "delayed constraint", it not only needs to pass up potential q_i results, but also negative events are passed up as we can't determine locally if they are in violation or not. The pseudo code of the nested execution algorithm is given in Figure 4. This function is called whenever a new event of the last positive event type in the outer query arrives. Figure 5 shows the algorithm for joining partial outer results with its children query results.

EXAMPLE 2. *Consider the query Q = SEQ(Recycle r, ! SEQ(Washing w, Drying dr, Sharpening s), Disinfection d, SEQ(Checking c, Relabeling rl), Operating op). When event instances of types Recycle, Washing, Drying, Sharpening, Disinfection, Checking, Relabeling and Operating arrive, they are pushed into their respective stacks. The outer query is first evaluated for a given window size followed by the inner sub-query. The outer query construction is triggered by the arrival of Operating events which are of the rightmost positive event type in the root query. For every partial result* $< r_i, d_j, op_k >$ *of the outer query SEQ(Recycle r, Disinfection d, Operating op), we compute the window constraints for its children queries. For details, see Figure 3. If we were to evaluate this query without predicates, all results for SEQ(Washing w, Drying dr, Sharpening s) and SEQ(Checking c, Relabeling rl) would be constructed for events that occur within [*r_i*.te,* d_j*.ts] and [*d_j*.te,* op_k*.ts], respectively. The outer operator joins with all results returned by its positive sub-query SEQ(Checking c, Relabeling rl). The outer result* $< r_i, d_j, op_k >$ *fails if results of the negative child query SEQ(Washing w, Drying dr, Sharpening s) exist. When evaluating Q with correlated predicates [id], the id is passed down from the outer query to the children sub-queries. Results involving events with the same id are constructed in the sub-queries.*

4. PERFORMANCE EVALUATION

The objective of our evaluation is to verify if our strategy gives the correct results so that they can be used as a benchmark to compare alternate future methods against. We verify using various types of queries. We also make note of the execution time to test the effectiveness and practicability of our method.

4.1 Experimental Setup

```
NestedExecution (query q_i, event e_i, Window W))
01 if(e_i triggers q_i result construction)
02 {Interval ts; ts_left=e_i.ts - W; ts_right=e_i.ts
   RecursiveCompute(q_i, e_i, ts)}
// compute q_i results
RecursiveCompute(query q_i, event e_i, ts)
01 finalResult fr[];
   buffers buf_children[];
02 result r[] = selfCompute( q_i , e_i);
03 if (q_i has no children queries)
04    {if(q_i ∈ labeledSubQueries (Sec  3.5))
05    return r[] with negative events in q_i;
06    else return r[]; }
07 else for each result r_j belongs to r[]
08       for each inner query child_j of q_i
09       Interval ts =
         IntervalConstraints(r_j, q_i.childj);
      // compute constraint window for each sub-expression
10       RecursiveCompute(q_i.child_j, e, ts);
11       for each inner query child_j of q_i
12       if (Eval(q_i, q_i.child_j, buf_children))
      // join positive children results

14       continue;
      // stop evaluation if a negative component is not empty.
15
```

Figure 4: Nested Execution Strategy

```
Eval (Query q_i, Query q_j, Buffer buf_children))
01 if (q_j ∈ labeledSubQueries)
02    tighten q_j results with negative events
03 if (q_j is a positive query in q_i)
04    join q_i and q_j results; return true;
05 else if(q_j.results are not empty)
      // q_j is a negative component
06    return false;
```

Figure 5: Result Evaluation

We have implemented our proposed nested query processing framework within the stream management system CHAOS [16] using Java. We ran the experiments on Intel Pentium IV CPU 2.8GHz with 4GB RAM. We evaluated our techniques using the real stock trades data from [17] with 10,000 event instances with a sliding window of size 10 ms.The data contained stock ticker, timestamp and price information.

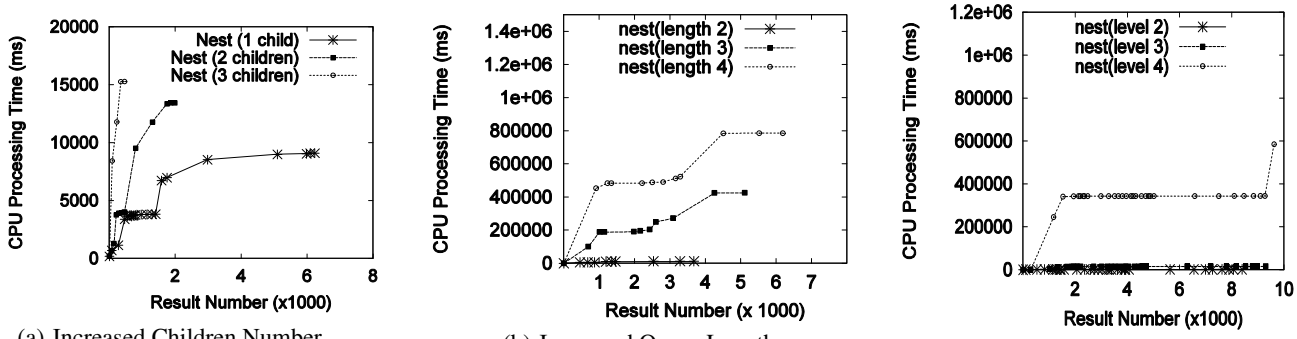
4.2 Varying Children Subquery Number

The first experiment processed queries with increased number of sub-queries from 1 to 3 (Figure 6(a)). q_3 generates minimum results using maximum processing time among the three queries. q_3 has more sub-queries to process which thus consumes more CPU processing time. Also, more outer SEQ(MSFT,ORCL,IPIX,INTC) results are filtered in q_3 as more constraints exist as compared to the other queries. As expected, the computation time increases with the number of sub-queries because the probability of finding patterns decreases with an increasing number of event types.

```
Increased Children Number:
q1=SEQ(MSFT,!SEQ(RIMM,AMAT),ORCL,IPIX,INTC);
q2=SEQ(MSFT,!SEQ(RIMM,AMAT),ORCL,!SEQ(YHOO,DELL),IPIX,INTC);
q3=SEQ(MSFT,!SEQ(RIMM,AMAT),ORCL,!SEQ(YHOO,DELL),IPIX,
        !SEQ(CSCO,QQQ),INTC);
```

4.3 Varying Subquery Lengths

The second experiment processed the queries below with increased sub-query lengths (from 2 to 4) as depicted in Figure 6(b).



(a) Increased Children Number

(b) Increased Query Length

(c) Increased Nesting Levels

Figure 6: Evaluating Nested Patterns

We observed that q_6 generates the most number of results and uses the most CPU processing time among the three queries. This is because q_6 includes the sub-query with the longest length which consumes more computational time. As expected, less outer SEQ(MSFT, ORCL,INTC) results are filtered in q_6 as the existence of a longer pattern is relatively less likely as compared to the other queries with shorter patterns within the same input stream.

```
Increased Query Length:
q4=SEQ(MSFT,!SEQ(RIMM,AMAT),ORCL,INTC);
q5=SEQ(MSFT,!SEQ(RIMM,AMAT,YHOO),ORCL,INTC);
q6=SEQ(MSFT,!SEQ(RIMM,AMAT,YHOO,DELL),ORCL,INTC);
```

4.4 Varying Subquery Nesting Levels

The third experiment processed the queries below with increased sub-query nesting levels as depicted in Figure 6(c). q_9 generates the most number of results and uses the most CPU processing time among the three queries. It is because q_9 includes the sub-query with the largest nesting levels which consumes more time to be computed. Less outer SEQ(MSFT, ORCL, INTC) results are filtered as it is relatively infrequent to have more events in levels occur in a sequence.

```
Increased Nesting Levels:
q7=SEQ(MSFT,!SEQ(IPIX,QQQ),ORCL,INTC);
q8=SEQ(MSFT,!SEQ(IPIX,SEQ(RIMM,AMAT),QQQ),ORCL,INTC);
q9=SEQ(MSFT,!SEQ(IPIX,SEQ(RIMM,SEQ(YHOO,DELL),AMAT),QQQ),
          ORCL,INTC);
```

5. NESTED QUERY OPTIMIZATION

Although the results of nested CEP queries obtained from the iterative execution strategy are correct, it produces results at a very slow rate which is attributed to the re-computation of the results for inner sub-queries every time an outer triggering event arrives which makes the processing expensive. To tackle this deficiency, we propose to cache and incrementally maintain the inner query results. Due to the sliding window, many intermediate results would continue to be valid from one sliding window to the next. Previously calculated results of the previous window should be cached and then be reused in the new window. In this paper we will only propose a direction for such an optimization technique. However this technique is not generic and cannot support negation or predicate correlation.

- **Cache Interval Extraction.** Assume $Q_i = SEQ(E_1, \ldots, E_i, SEQ(E_{i+1}, \ldots, E_{i+j}), E_{i+j+1}, \ldots, E_n)$. For a given triggering event $e_n \in E_n$, the left bound of the interval attached to the subexpression SEQ(E_{i+1}, …, E_{i+j}) is given by e_i.ts such that e_i has the minimum timestamp among all events of type E_i which have arrived so far. Similarly, the right bound of the interval is given by an event e_{i+j+1}.ts such that e_{i+j+1} has the maximum timestamp among all events of type E_{i+j+1} which have arrived so far. The extracted interval is attached to each cache representing the valid time period for the cached results.

- **Interval-driven Cache Expansion.** We update the cache content when a new triggering event e_t arrives. That is, given a new triggering event instance e_i, we calculate the new cache interval. For each subexpression, we compare the interval [i, j] attached to the cache to the new interval [m, n]. By the way our algorithm works, $i = m$, since the left bound is maintained at the event with minimum timestamp. We compute the sub-query SEQ($E_{i+1}, \ldots, E_{i+j}$) for all triggering events e_{i+j} between the interval [j, n] New results are appended to the cache for each subexpression triggered by events occurring between the right bounds of [j, n].

- **Interval-driven Cache Reduction.** When a triggering event e_t arrives, events with timestamp less than e_t - window are purged from their stacks. Similarly, caching results involving events with timestamp less than e_t - window are deleted from the cache as the window constraint will be violated if these results join with the new triggering event e_t in the final result.

EXAMPLE 3. *In Figure 7, when the triggering event o_{26} arrives, it is inserted into the Operating stack and triggers execution. [1, 15] and [8, 26] are extracted time intervals for the subexpressions SEQ(Washing, Drying, Sharpening) and SEQ(Checking, Relabeling), respectively. SEQ(Washing, Drying, Sharpening) results are constructed based on all events that occurred during [1, 15]. Similarly, SEQ(Checking, Relabeling) events occurring during [8, 26] are constructed and cached. When the new triggering event o_{30} arrives, we determine the interval for SEQ(Washing, Drying, Sharpening) is still [1, 15]. Thus the cache is still complete and thus we can reuse results in the cache. For subexpression SEQ(Checking, Relabeling), we find the new interval [8, 30] overlaps with the previous interval [8, 26]. Conceptually, we could reuse the caching results related to [8, 26] and we must compute the new additions to our cache. New SEQ(Checking,Relabeling) results are triggered by Relabeling events occurring between [26, 30] such as rl_{28}. Assume the window size is 30. When o_{34} arrives, all caching results involving primitive events with time-stamp less than 4 expire. So $< w_2, dr_6, s_7 >$, $< w_2, dr_3, s_7 >$ etc are deleted from the cache. The meta-data attached to the cache for SEQ(Washing, Drying, Sharpening) is updated from [1, 15] to [4, 15].*

5.1 Evaluating Optimized Nested Execution: Caching Results

We process query q_{10} comparing the optimized execution by the caching technique to the one without caching as in Figure 8. Caching helps in avoiding repeated computation for the subquery SEQ(QQQ,AMAT,DELL) as our results demonstrate. Clearly, we

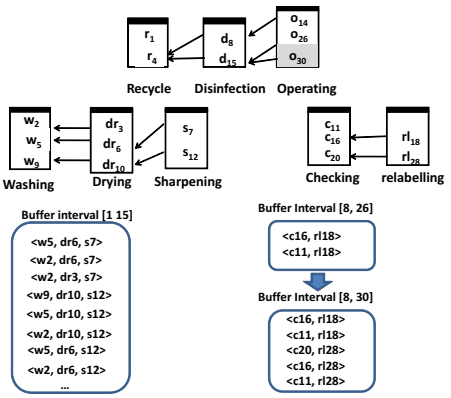


Figure 7: Interval Driven Subexpression caching

will have different gain with different reuse opportunities which may be caused by larger windows, more expensive sub-queries, etc.

```
Increased Nesting Levels:
q10 = SEQ(YHOO,SEQ(QQQ,AMAT,DELL),ORCL,IPIX);
```

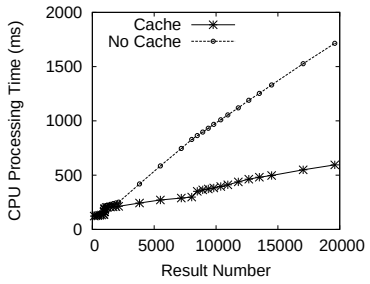


Figure 8: Interval-Driven Caching

6. RELATED WORK

The existing CEP systems [1, 2, 3, 6] do not focus on the execution of nested sequence queries as tackled here. The query language of the CEDR [6] system supports nested sequence queries. However, the execution strategy for nested queries is not given.

Complex queries used in decision support applications often have multiple correlated sub-queries and table expressions, possibly across several levels of nesting. It is usually inefficient to directly execute a correlated query. Consequently, algorithms such as magic decorrelation [18] and complex query decorrelation [19] have been proposed to decorrelate the query. However, existing decorrelation algorithms deal with only relational queries, that is, these algorithms are neither described nor tested in the CEP streaming context.

For SQL queries, [20] discusses whether a query result should be admitted to the cache and which results are to be purged in the static data context. In semantic caching [21], a semantic description of the data in a cache is maintained which allows for a compact specification. Semantic descriptors have also be shown to be of importance for query caching in the XML context [22, 23, 24]. However, sophisticated cache matching algorithms had to be designed to deal with query containment, namely, with extracting related XQuery subexpressions possibly with alternate hierarchical XML structures yet the same content [22].

7. CONCLUSION

In this paper, we introduced a comprehensive iterative execution strategy for processing nested CEP queries. An algebraic query plan for the execution of nested CEP queries was designed. We then developed a window constraint tightening technique to correctly process sub-queries. We also presented execution strategies for handling predicates in nested queries. Optimization using interval driven cache expansion and reduction was introduced. We plan to study additional optimization techniques in the future.

8. ACKNOWLEDGEMENTS

This work is supported by HP Labs Innovation Research Program and National Science Foundation under grants NSF IIS 0917017. Ismail Ari is supported by TUBITAK Grant 109E194. We thank Database System Research Group at WPI for valuable comments.

9. REFERENCES

[1] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams." in *SIGMOD Conference*, 2006, pp. 407–418.

[2] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White, "Cayuga: A general purpose event monitoring system." in *CIDR*, 2007, pp. 412–422.

[3] Y. Mei and S. Madden, "Zstream: a cost-based query processor for adaptively detecting composite events," in *SIGMOD Conference*, 2009, pp. 193–206.

[4] J. M. Boyce and D. Pittet, "Guideline for hand hygiene in healthcare settings," *MMWR Recomm Rep.*, vol. 51, pp. 1–45, 2002.

[5] "Wireless sensor networks for home health care," pp. 832–837, 2007.

[6] R. S. Barga, J. Goldstein, M. Ali, and M. Hong, "Consistent streaming through time: A vision for event stream processing." in *CIDR*, 2007, pp. 363–374.

[7] M. Liu, E. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, and A. Mehta, "Nested complex event processing for real-time event analytics," in *BIRTE*, 2010.

[8] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim, "Composite events for active databases: Semantics, contexts and detection." in *VLDB*, 1994, pp. 606–617.

[9] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White, "Cayuga: A general purpose event monitoring system." in *CIDR*, 2007, pp. 412–422.

[10] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *SIGMOD Conference*, 2008, pp. 147–160.

[11] D. Gyllstrom, J. Agrawal, Y. Diao, and N. Immerman, "On supporting kleene closure over event streams," in *ICDE*, 2008, pp. 1391–1393.

[12] P. Seshadri, H. Pirahesh, and T. Y. C. Leung, "Complex query decorrelation," in *ICDE*, 1996, pp. 450–458.

[13] I. S. Mumick, S. J. Finkelstein, H. Pirahesh, and R. Ramakrishnan, "Magic is relevant," in *SIGMOD Conference*, 1990, pp. 247–258.

[14] E. Wong and K. Youssefi, "Decomposition - a strategy for query processing," *ACM Trans. Database Syst.*, vol. 1, no. 3, pp. 223–241, 1976.

[15] J. M. Smith and P. Y.-T. Chang, "Optimizing the performance of a relational algebra database interface," *Commun. ACM*, vol. 18, no. 10, pp. 568–579, 1975.

[16] C. Gupta, S. Wang, I. Ari, M. Hao, U. Dayal, A. Mehta, M. Marwah, and R. Sharma, "Chaos: A data stream analysis architecture for enterprise applications," in *CEC'09*, 2009, pp. 33–40.

[17] "I. inetats. stock trade traces. http://www.inetats.com/."

[18] C. Beeri and R. Ramakrishnan, "On the power of magic," *J. Log. Program.*, vol. 10, no. 1/2/3&4, pp. 255–299, 1991.

[19] P. Seshadri, H. Pirahesh, and T. Y. C. Leung, "Complex query decorrelation," in *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*. IEEE Computer Society, 1996, pp. 450–458.

[20] J. Shim, P. Scheuermann, and R. Vingralek, "Dynamic caching of query results for decision support systems," in *SSDBM*, 1999, pp. 254–263.

[21] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan, "Semantic data caching and replacement," in *VLDB*, 1996, pp. 330–341.

[22] L. Chen and E. A. Rundensteiner, "Xquery containment in presence of variable binding dependencies," in *WWW*, 2005, pp. 288–297.

[23] L. Chen, E. Rundensteiner, and S. Wang, "Xcache: A semantic caching system for xml queries," in *ACM SIGMOD*, 2002, pp. 618–618.

[24] L. Chen, S. Wang, and E. A. Rundensteiner, "Replacement strategies for xquery caching systems," in *Data and Knowledge Engineering Journal*, 2004, pp. 145–175.

Query-Driven Data Collection and Data Forwarding in Intermittently Connected Mobile Sensor Networks

Wei Wu
School of Computing
National University of Singapore
wuw@nus.edu.sg

Hock Beng Lim
Intelligent Systems Centre
Nanyang Technological University
limhb@ntu.edu.sg

Kian-Lee Tan
School of Computing
National University of Singapore
tankl@comp.nus.edu.sg

ABSTRACT

In sparse and intermittently connected Mobile Sensor Networks (MSNs), the base station cannot easily get the data objects acquired by the mobile sensors in the field. When users query the base station for specific data objects, the base station may not have received the necessary data objects to answer the queries. In this paper, we propose to use a Mobile Data Collector (MDC) to collect the data objects from the mobile sensors that the base station needs for answering queries. To facilitate the MDC's data collection, we design a location-based data forwarding protocol that exploits the location metadata of data objects and uses caching to improve data availability in the MSNs. Results of performance study show that our solution can reduce query response times on the base station.

1. INTRODUCTION

Mobile sensor networks (MSNs) are very useful in reconnaissance, disaster rescue, and environment monitoring tasks. For examples, mobile sensors can be used to gather information in battlefields and earthquake areas.

In most applications of MSNs, users (such as commander or rescue personnel) will want to access the data objects acquired by the mobile sensors. They query the base station for the data objects they want. Answering these queries timely is very desirable (sometimes even critical).

Unfortunately, in sparse MSNs where the sensors and the base station are only intermittently connected, many queries may only be answered after a quite long time. Due to the intermittent connections, it is difficult for the mobile sensors to send data objects to the base station and for the base station to pull data objects from the sensors.

Mobile data collectors (MDCs) [4], [10], [12] [3] are widely used to collect data objects in sensor networks. A MDC collects data objects by moving in the sensor networks, getting data objects from sensors within wireless communication range, and moving back to the base station.

In this paper, we propose to use a MDC to do *query-driven* data collection in sparse MSNs to reduce the average query answering time. When the base station receives a query but does not have the data object that the query requests for, it lets the MDC to collect the data object. We focus on data collection for answering *spatial queries* that explicitly request for data objects that are acquired at specific locations by the mobile sensors.

The challenge of query-driven data collection in intermittently connected MSNs is that the base station and the MDC do not know which mobile sensor has the data object they want. Due to disconnections, they cannot simply flood a message to all the sensors and find this out.

We design a query-driven data collection solution called `F4C` (Forwarding for Collection). Our main idea is to use spatial information of data objects to direct data forwarding and use spatial information of queries to direct data collection so that the MDC will have a good chance of meeting a mobile sensor that carries the data object that it needs.

In `F4C`, when the mobile sensors forward a data object acquired at location l to the base station, they keep (when possible) that data object in a region along the shortest physical path from l to the base station. When a MDC needs to collect a data object that was acquired at l, it simply moves towards l along the shortest path from the base station to l. The mobile sensors make data forwarding decisions that reduce the distance the MDC needs to move before it can get the data it wants. Furthermore, caching is used to increase the data availability among the mobile sensors. Through results from simulation, we show that `F4C` can reduce the average query answering time on the base station.

The remainder of the paper is organized as follows. We describe the system model in Section 2, and present `F4C` in Section 3. Results of performance study are shown in Section 4. Related work is briefly discussed in Section 5. We conclude this paper and list the directions of future work in Section 6.

2. SYSTEM MODEL

2.1 Mobile Sensor Network (MSN)

The system consists of a stationary base station BS, n mobile sensors (s_1,s_2,...,s_n), and a mobile data collector (MDC) in a task field A. They use wireless technology such as WiFi for communication. Two nodes (we use "node" to refer to a sensor, the BS, or a MDC) can communicate *directly* only if the distance between them is smaller than the wireless technology's communication range r.

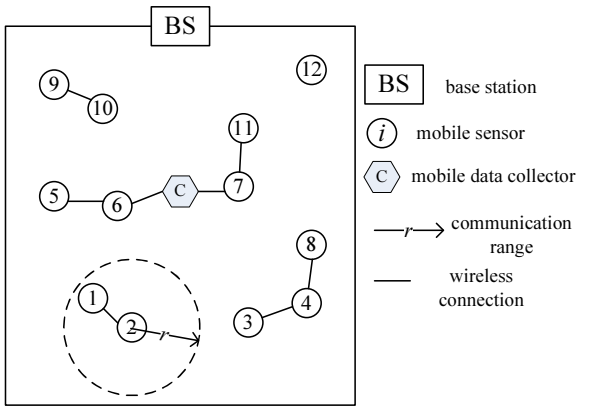


Figure 1: Example of a sparse MSN where a MDC is used to do data collection.

The mobile sensors move in the task field following a certain mobility model. Their move speed is v. We assume that all mobile nodes have GPS equipped so they always know their own locations. The BS is stationary and its location is known to all mobile nodes.

This work assumes a mobile sensor network that is sparse and only intermittently connected due to low sensor density (and/or short communication range) and sensor movement. Figure 1 shows an example of an intermittently connected MSN where there are twelve mobile sensors.

2.2 Data Objects

The mobile sensors acquire data objects when they move in the task field. The location where a data object is acquired is kept as a part of the data object's metadata. We use D_p to refer to a data object that is acquired at location p. In addition, a spatial region is also associated with the data object. It is determined by the location p and the sensor's sensing range, and is the geographical region whose feature is captured in the data object. For example, if the data object is an image, then the spatial region associated with it is the area captured in the image.

2.3 Queries

Users of the system query the base station for data objects by spatial predicates. For simplicity, we assume each query asks for one data object acquired at a specific location. We use *query location* to refer to the location specified in a query.

A data object can be used to answer a query if the data object's spatial region covers the query location. The time duration from the base station gets a query to the base station answers the query is the query's *response time*.

2.4 Data Forwarding

The mobile sensors always try to forward their data objects to the base station. They forward data objects in a carry-and-forward fashion [2], because the sensor network is only intermittently connected. When a sensor acquires a data object or receives a data object from a neighbor (a *neighbor* refers to a node within communication range) but has no suitable neighbor to forward it to, the sensor carries the data object and tries to forward it later.

In sparse MSNs, the sensors have only limited communication opportunities to forward data objects. For this reason, we assume that a mobile sensor does not forward a data object multiple times.

The mobile sensors make their data forwarding decisions based on a data forwarding algorithm. We will describe our location-based data forwarding algorithm in Section 3.

2.5 Data Collection

The MDC's job is to collect data objects from the mobile sensors. When there is no pending queries on the base station, the MDC moves in the task field to collect data objects from mobile sensors and periodically returns to the BS. After returning to the BS, the MDC sends the data that it has collected to the BS.

If the BS has pending queries when the MDC returns to it, the BS sends a query to the MDC and lets it collect a data object for the query. We call this *query-driven data collection*, and use *mission query* to refer to the query that the base station sends to the MDC.

In this paper we look at the problem of data collection for *one* query. In future work, we will study the problem of data collection for *multiple* queries.

We assume that the MDC also has sensing capability. If the MDC failed to get a data object from the sensors for the mission query, it can move to the query location to acquire a data object for the query.

3. F4C: FORWARDING FOR COLLECTION

`F4C` (Forward for Collection) is specially designed for query-driven data collection in intermittently connected MSNs. Its goal is to reduce the distance that the MDC needs to move before it gets a data object for the mission query.

We define two terms in `F4C`: a data object's *collection path* and *forwarding region*. The MDC collects a data object by moving on the data object's collection path. The forwarding region is an area along the collection path. The mobile sensors keep a data object in its forwarding region when they forward the data object towards the base station.

3.1 Collection Path and Forwarding Region

For a data object D_p acquired at location p, we define the shortest physical path in the field from the base station to p as D_p's *collection path*, and the union of the points in the field whose distances to D_p's *collection path* are shorter than r (the wireless communication range) as D_p's *forwarding region*. We will use $Path(D_p)$ to denote D_p's collection path, and $Region(D_p)$ to denote D_p's forwarding region.

Figure 2 shows an illustration of a data object's collection path and forwarding region in a field where there are no obstacles. If there are obstacles in the filed, the collection path may not be a straight line. For simplicity, in this paper we will use examples where there are no obstacles in the field. Note that `F4C` also applies to fields where there are obstacles.

Note that D_p's collection path is determined by the location of BS and p (the location metadata of D_p), and D_p's forwarding region is determined by its collection path and the wireless communication range r. The idea is that when the MDC moves along D_p's *collection path* it will encounter the mobile sensors in D_p's *forwarding region*.

A data object's collection path and forwarding region are fixed and they are independent of the mobile sensor that is currently carrying it. Since the location of the BS is known

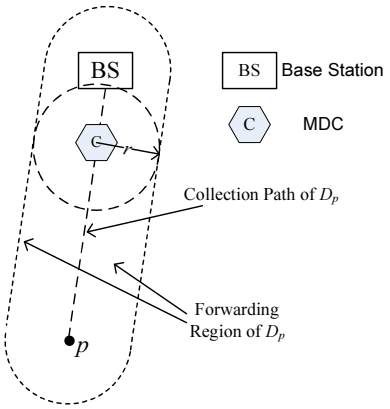


Figure 2: Illustration of a data object's Collection Path and Forwarding Region.

to all the mobile nodes, and data objects have location metadata, the mobile sensors can compute their data objects' collection paths and forwarding regions by themselves. Given a query, the MDC can also immediately compute the collection path of the data object that the query requests for.

3.2 Query-Driven Data Collection

When the BS has a pending query that requests for a data object acquired at location l and the MDC is connected to the BS, the BS sends the query to the MDC and lets it collect a data object for the query.

In `F4C`, the MDC's process of query-driven data collection is very simple. It simply moves from the BS towards l on $Path(D_l)$. When the MDC encounters a mobile sensor, it queries the mobile sensor for a data object that can answer the mission query. If the mobile sensor has such a data object, it sends the data object to the MDC. After receiving the data object, the MDC moves back to the BS. If none of the sensors that the MDC encountered has a data object that is an answer to the mission query, the MDC will arrive at l. The MDC acquires a data object by itself at l, and moves back to the BS.

Note that although the MDC's main task in a query-driven data collection mission is to collect a data object for the mission query, it also collects other data objects when it encounters the mobile sensors. We will elaborate on this in Section 3.4.2.

The time cost of collecting a data object for a query is roughly twice the time from it got the query to it gets a data object that can answer the query. In the worst case, the MDC arrives at the query location and then acquires such a data object. Reducing the distance that the MDC needs to move before it gets a data object for the mission query will be an effective way to reduce the time cost of query-driven data collection.

3.3 Location-based Data Forwarding

The general idea of data forwarding in `F4C` is keeping the data objects in their own forwarding regions when the mobile sensors forward the data objects towards the base station. The objective is to maintain a data object's availability in its forwarding region so that the MDC can easily get the data object by moving on its collection path.

Suppose sensor s_i currently carries a data object D_p. s_i computes D_p's forwarding region using D_p's location metadata and the location of the BS (which is known to all the sensors). s_i knows its own location and exchanges location information with neighbors periodically. s_i makes different forwarding decisions for D_p based on (1) whether it is inside D_p's forwarding region and (2) neighbors' location.

- If s_i is in D_p's forwarding region, s_i forwards D_p to a sensor that is also in D_p's forwarding region but is nearer to the base station.
- If s_i is not in D_p's forwarding region, s_i forwards D_p to a sensor that is in D_p's forwarding region. If none of s_i's neighbors is in D_p's forwarding region, s_i forwards D_p to a sensor that is closer to D_p's forwarding region.

In both cases, if none of s_i's neighbors satisfies the conditions, s_i carries the data object and tries to forward it later.

3.4 Prioritize Data Objects in Forwarding

In sparse MSNs, each mobile sensor will be carrying many data objects, because the sensor keeps acquiring data objects in the field but has limited opportunities to forward data objects to the BS or the MDC. Therefore, when a sensor encounters a neighbor, it will have more data objects than it can send to the neighbor during the short connection duration with the neighbor. The sensor has to decide what data objects should be forwarded to the neighbor. Based on whether the neighbor is a mobile sensor or the MDC, different algorithms are used to select the data objects for forwarding. Both algorithms are optimized to help reduce the distance that the MDC needs to move before getting a data object for the mission query.

3.4.1 Forwarding to a Neighboring Sensor

To help the sensors decide which sensor should carry what data object, we define a measure called *collection-distance* and let the sensors make data forwarding decisions based on their collection-distance of the data objects. Given a data object, a sensor's collection-distance is the distance that a DMC needs to move to get the data object if the sensor carries that data object.

Let $Circle(s_i, r)$ denote the circle centered at the location of s_i with radius r which is the wireless communication range. Given a data object D_p, if $Circle(s_i, r)$ does not intersect with $Path(D_p)$, s_i's collection-distance for D_p is defined as the length of $Path(D_p)$; otherwise, let I be the intersection of $Circle(s_i, r)$ and $Path(D_p)$ that is closer to the BS, s_i's collection-distance for D_p is the distance from the BS (along $Path(D_p)$) to I.

Figure 3 illustrates the definition of collection-distance with two examples. $Circle(s_1, r)$ does not intersect with $Path(D_p)$, so s_1's collection-distance for D_p is the length of $Path(D_p)$. $Circle(s_2, r)$ intersects with $Path(D_p)$ at I_1 and I_2. I_2 is closer to the BS, so s_2's collection-distance for D_p is the distance from the BS to I_2 along $Path(D_p)$.

Intuitively, a sensor's collection-distance for a data object is the distance that the MDC needs to move alone the data object's collection path before it can get the data object from the sensor. When a sensor is outside a data object's forwarding region, its collection-distance for the data object is the length of the data objet's collection path.

Given two sensors s_i and s_j, a data object D_p, and let $cd(s_i, D_p)$ and $cd(s_j, D_p)$ denote s_i and s_j's collection-distances

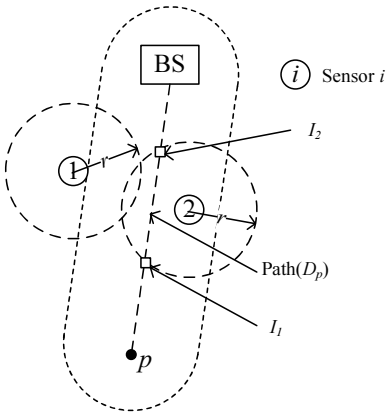


Figure 3: Illustration of forward distance definitions.

for D_p, we define ($cd(s_i, D_p)$ - $cd(s_j, D_p)$) as the *delta-collection-distance* between s_i and s_j for D_p.

When s_i encounters s_j, s_i uses delta-collection-distance for the data objects (that s_i carries) to prioritize the forwarding of the data objects. First, the data objects whose delta-collection-distances are positive are considered for forwarding. s_i keeps sending to s_j the data object for which the delta-collection-distance between s_i and s_j is the largest. This process goes on until s_i and s_j are not connected any more or no data object has a positive delta-collection-distance.

For the data objects whose delta-collection-distance is zero, the ones for which s_i is outside their forwarding regions are considered for forwarding. Since s_i and s_j have the same collection-distance for these data objects, s_j must also be outside the objects' forwarding regions. s_i forwards a object to s_j if s_j is closer to its forwarding region. Due to space limitation, we do not further elaborate on this.

Note that we are considering single-path forwarding where a sensor forwards a data object only once. Once s_i sends a data object to s_j, s_i may remove the data object from its storage. Also note that s_j may also forward data objects to s_i. Our design is at the application layer and assume that the allocation of communication slots is controlled by a MAC (Media Access Control) layer protocol.

3.4.2 Forwarding to the MDC

During the MDC's query-driven data collection missions, the MDC collects not only data objects for the mission queries but also other data objects. When the MDC encounters a mobile sensor, the mobile sensor can send data objects to the MDC as long as they are connected.

A mobile sensor prioritizes the data objects for forwarding to the MDC by the lengths of their collection paths. The data object whose collection path is the longest gets first forwarded to the MDC. For example, suppose sensor s_i carries data object D_a and D_b, and the collection path of $Path(D_b)$ is longer than $Path(D_a)$, and s_i can only forward one data object to the MDC due to limited connection time. s_i will forward D_b to the MDC.

The rationale behind this design is that the data objects with longer collection paths are more difficult for the MDC to collect if there is a query in the future requesting for it. Recall that in the worst case the MDC has to move to the query location to acquire a data object for the query. By forwarding the data objects with longer collection paths to the MDC, the base station will get (from the MDC) these data objects and will be able to answer the queries that request for such data objects. The queries requesting for data objects closer to the BS are easier to be answered because it is easier for the MDC to collect these data objects even if they are not in their forwarding regions.

3.5 Caching

In `F4C`, the sensors do their best to keep a data object in its forwarding region, but sometimes the sensor carrying the data object may move out from the data object's forwarding region and none of its neighbors is in the data object's forwarding region. To improve the chance that the MDC encounters a sensor that has the data object which the MDC is looking for, we use caching to further improve the data availability among the mobile sensors.

After a sensor s_i forwards a data object to a neighboring sensor s_j, s_i does not delete the local copy of the data object but keeps it as a caching in local storage. s_i will not forward the copy of the data object to any other sensor (because we are considering single-path data forwarding rather than multiple-path data forwarding), but can send it to the MDC if the MDC needs this data object for answering its mission query. Recall that when the MDC encounters a sensor, the MDC will check whether the sensor has a data object for the query.

4. PERFORMANCE STUDY

We study the performance of `F4C` through simulation. The aim of the experiments is to investigate whether `F4C` can help reduce the average query answering time at the base station and how the system parameters (such as the number of sensors, size of data object, etc) will affect `F4C`'s performance.

Since we are not aware of any existing query-driven data collection scheme for sparse MSNs, we compare `F4C` to a solution where the MDC moves towards the query location (as in `F4C`) and the mobile sensors use geographical greedy routing [9] in data forwarding. We choose geographical greedy routing for comparison because it has been regarded as a very effective data forwarding algorithm in sensor network. In experiments we call this method GeoGreedy. In GeoGreedy, only the locations of neighbors are considered and a sensor always forwards data objects to a neighbor that is closer to the BS (no matter whether the neighbor is in the data objects forwarding regions).

We designed a simulation package for mobile sensor networks and implemented it in Java. In our simulation experiments, n mobile sensors are initially randomly placed in a 600 Meters * 600 Meters field and they move in the field at speed v according to a random waypoint mobility model. The move speed of the MDC is $2 * v$. Each sensor acquires a data object every T_s seconds. The size of a data object is D KB. The BS receives a query every T_q seconds. 100 queries with random query locations in the field are issued to the base station. The wireless communication bandwidth is 2Mbps and the communication range is r Meters. Table 1 lists the parameters and their values.

The performance measure is the average query answering time on the base station. Both the queries that are answered by the base station right away and the queries answered through the MDC's query-driven data collection are accounted for in the calculation of the average query

Table 1: System Parameters

Parameter	Unit	Default	Range
number of sensors n		30	20 - 50
move speed v	Meters/s	2	1 - 8
data size D	KB	500	100 - 1000
sense interval T_s	seconds	20	10 - 60
query interval T_q	seconds	20	10 - 60
communication range r	Meter	100	50 - 150

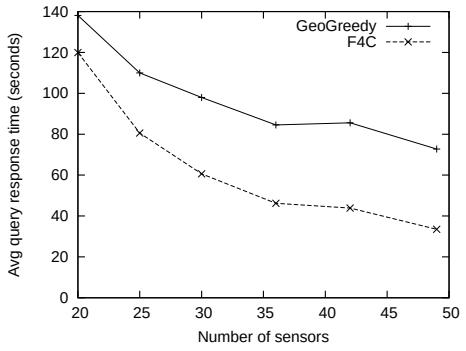


Figure 4: E ect of the number of sensors on average query response time.

answering time.

In each set of experiments we vary the value of one parameter and study its effect on F4C and GeoGreedy. Due to space limitation, we will present only some representative experiment results.

4.1 Effect of the Number of Sensors

The number of mobile sensors immediately affect the density of sensors in the field and the connectivity of the sensor network. Figure 4 shows the effect of the number of sensors on the performance of F4C and GeoGreedy. We see that F4C reduces the average query response time and the improvement over GeoGreedy is between 15% to 50%. When there are very few (e.g., 20) sensors moving in the field, the sensor network is very sparse and most of the times a sensor has no neighbor to forward its data objects to. The sensors cannot keep data objects in their forwarding regions and the MDC needs to move to the query locations to get data objects for most of the mission queries. As the number of sensors increases, F4C can effectively direct the sensors to forward data objects to neighbors in data objects' forwarding regions, so F4C starts to clearly outperform GeogGreedy.

F4C outperforms GeoGreedy because in F4C a data objects' availability along its collection path is generally higher than that in GeoGreedy.

4.2 Effect of Sensors' Move Speed

The effect of sensors' move speed on the performance of F4C and GeoGreedy is shown in Figure 5. We observe that F4C consistently outperform GeoGreedy, and the average query answering times decrease as the sensors' move speed increases. Sensors' move speed affects the number of neighbors that a sensor will encounter during a period of time. When the sensors and the MDC move at a faster speed, they encounters new neighbors more frequently but have shorter connection time with the neighbors, and the MDC

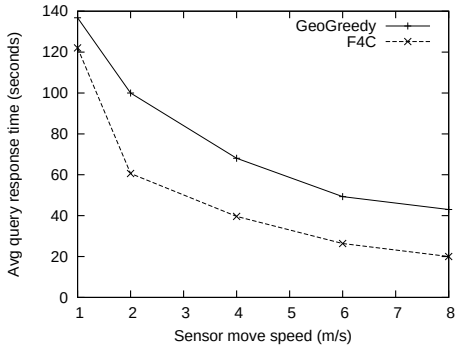


Figure 5: E ect of sensor's move speed on average query response time.

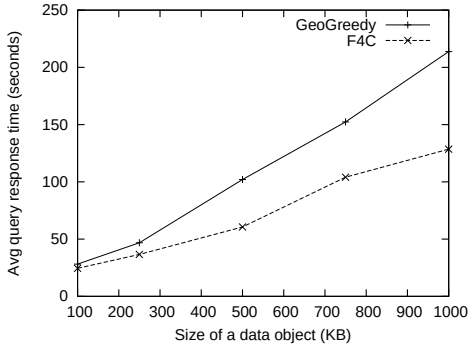


Figure 6: E ect of data object size on average query response time.

can arrive at the query locations in shorter time. In F4C, the mobile sensors exploit the encounters and keep data objects in their forwarding regions by sending carefully selected (through prioritization) data objects to the neighbors.

4.3 Effect of the Size of a Data Object

Figure 6 shows how the size of a data object will affect the performance of F4C and GeoGreedy. The average query answering times in both F4C and GeoGreedy are longer when the data objects are bigger. This is because as the data object size increases a sensor can forward a smaller number of data objects to a neighbor during their limited connection time. This not only means that a smaller number of data objects can be kept in their forwarding regions but also means that the MDC will receive a smaller number of data objects from the sensors. As a result, the BS will get less data objects from the MDC and more queries need to be answered through the MDC's data collection in the field.

4.4 Effect of Other Parameters

Due to space limitation, we will not present in detail the experiment results on communication range, sense interval, and query interval. All experiment results show that F4C results in shorter average query answering time when compared to GeoGreedy. Longer communication range makes the sensor network better connected and lets the nodes have more time for communication, and thus has a positive effect on average query answering times. Longer sense interval and longer query interval also have positive effects on average query answering times. Longer sense interval means

the sensors will gather smaller amount of data and it will be easier for them to keep the data objects in forwarding regions or send to the MDC. Longer query intervals means that the BS will receive more data before it receives a new query so it is more likely that the query can be answered right away.

5. RELATED WORK

Several routing protocols designed for mobile ad-hoc networks (MANET) and wireless sensor networks make use of mobile nodes' location information. The well-known examples include *compass routing* [7], DREAM (distance routing effect algorithm for mobility) [1], LAR (location-aided routing) [6], GPSR (greedy perimeter stateless routing) [5], and GEAR (Geographical and Energy Aware Routing) [11]. [9] investigated the performance of geographic greedy routing algorithms in sensing-covered networks and showed that simple greedy geographic routing is an effective routing scheme in many sensing-covered networks.

The data forwarding method proposed in this paper differs from existing location-based routing protocols in that it exploits not only the location of the mobile nodes but also the location information of the data objects. Furthermore, our data forwarding method is specially designed to facilitate a MDC's query-driven data collection.

Studies on mobile data collectors in sensor networks are also related to this work. In existing work on mobile data collection [8], [13], [4], [10], [12] [3], however, the focus is to minimize the energy consumption of the sensors. The mobile data collectors in existing work either have fixed collection routes or move randomly in the field. The main difference between this work and existing work on mobile data collectors is that we focus on data collection for query answering on the base station and use query location to direct the movement of the mobile data collectors.

6. CONCLUSION AND FUTURE WORK

In sparse and intermittently connected mobile sensor networks, the base station cannot easily get data objects from the mobile sensors to answer the queries received from the users. We propose to use a mobile data collector (MDC) to do query-driven data collection so that the base station can answer the queries more timely. We present a data forwarding and data collection solution called `F4C` (Forwarding for Collection) that reduces the time that the MDC needs to move before it gets data objects for queries. In `F4C`, a MDC collects a data object for a query by simply moving from the BS towards the query location, and the mobile sensors keep data objects available in regions along the data objects' collection paths. The mobile sensors make data forwarding decisions based on their locations and the data objects' location metadata. The algorithms that the mobile sensors use to prioritize the data objects for forwarding are designed to help reduce average collection time. We did simulation to study the performance of our proposed solution. Experiment results show that `F4C`can help reduce average query processing time on the base station.

This preliminary work only considers spatial information of the query and data objects. A future work is to take query and data objects' temporal information also into account. In `F4C`, single-path data forwarding is assumed. Another direction of future work, therefore, is to consider multi-path forwarding and study the trade-off between data availability and data traffic. Last but not the least, we believe that it will be interesting to design an algorithm for the MDC to collect data for multiple queries.

7. ACKNOWLEDGMENTS

This work is partially supported by a research grant R-252-000-352-232 (TDSI/08-001/1A) from the Temasek Defense Systems Institute.

8. REFERENCES

[1] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward. A distance routing effect algorithm for mobility (dream). In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 76–84, New York, NY, USA, 1998. ACM.

[2] K. W. Chen. Cafnet: A carry-and-forward delay-tolerant network. MIT Master Thesis, 2007.

[3] M. D. Francesco, K. Shah, M. Kumar, and G. Anastasi. An adaptive strategy for energy-efficient data collection in sparse wireless sensor networks. In *EWSN*, pages 322–337, 2010.

[4] Y. Gu, D. Bozdağ, R. W. Brewer, and E. Ekici. Data harvesting with mobile elements in wireless sensor networks. *Comput. Netw.*, 50(17):3449–3465, 2006.

[5] B. Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MOBICOM*, pages 243–254, 2000.

[6] Y.-B. Ko and N. H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wirel. Netw.*, 6(4):307–321, 2000.

[7] E. Kranakis, S. O. C. Science, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *in Proc. 11 th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[8] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: modeling a three-tier architecture for sparse sensor networks. In *IEEE SNPA Workshop*, pages 30–41, 11 May 2003.

[9] G. Xing, C. Lu, R. Pless, and Q. Huang. On greedy geographic routing algorithms in sensing-covered networks. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 31–42, New York, NY, USA, 2004. ACM.

[10] G. Xing, T. Wang, W. Jia, and M. Li. Rendezvous design algorithms for wireless sensor networks with a mobile base station. In *MobiHoc '08: Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 231–240, New York, NY, USA, 2008. ACM.

[11] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks. Technical report, 2001.

[12] M. Zhao and Y. Yang. Bounded relay hop mobile data gathering in wireless sensor networks. In *MASS*, 2009.

[13] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc*, pages 187–198, New York, NY, USA, 2004. ACM.

DEMS: A Data Mining Based Technique to Handle Missing Data in Mobile Sensor Network Applications

Le Gruenwald Md. Shiblee Sadik Rahul Shukla Hanqing Yang
School of Computer Science
University of Oklahoma
Norman, Oklahoma, USA
{ggruenwald, shiblee.sadik, rahul.shukla-1, hqyang3}@ou.edu

ABSTRACT
In Mobile Sensor Network (MSN) applications, sensors move to increase the area of coverage and/or to compensate for the failure of other sensors. In such applications, loss or corruption of sensor data, known as the missing sensor data phenomenon, occurs due to various reasons, such as power outage, network interference, and sensor mobility. A desirable way to address this issue is to develop a technique that can effectively and efficiently estimate the values of the missing sensor data in order to provide timely response to queries that need to access the missing data. There exists work that aims at achieving such a goal for applications in static sensor networks (SSNs), but little research has been done for those in MSNs, which are more complex than SSNs due to the mobility of mobile sensors. In this paper, we propose a novel data mining based technique, called Data Estimation for Mobile Sensors (DEMS), to handle missing data in MSN applications. DEMS mines the spatial and temporal relationships among mobile sensors with the help of virtual static sensors. DEMS converts mobile sensor readings into virtual static sensor readings and applies the discovered relationships on virtual static sensor readings to estimate the values of the missing sensor data. We also present the experimental results using both real life and synthetic datasets to demonstrate the efficacy of DEMS in terms of data estimation accuracy.

Keywords
Sensors, Missing Data, Mobile Sensor Networks

1. INTRODUCTION
A wireless sensor network (WSN) can be defined as a set of independent sensors which can solve cooperatively some monitoring based applications [1]. Typical applications of WSN include environmental monitoring [2], scientific investigation [3], civil structure flaw detection, battle surveillance and medical applications [4]. However, successful monitoring of any physical phenomenon is directly dependent on the appropriate deployment of the sensors [5], [6]. In a static sensor network (SSN), the sensors' positions remain stationary after the initial deployment. In addition, the areas covered by the sensors are dependent on the initial network configuration and remain unchanged over time [7]. An inappropriate deployment of sensors in a SSN may partition the monitoring area into regions either covered by at least one sensor and/or devoid of any sensors [7]. Therefore, while a covered region may be monitored by unnecessary multiple sensors, the regions uncovered by sensors may not be monitored at all leading to inaccurate results. Also, certain restrictions, such as hostile environments and disaster areas [8], make initial, manual deployment of sensors impossible. Finally, certain applications like monitoring atmosphere or ocean environment require constant mobility that can be achieved only if the sensors themselves are mobile [7]. Consequently, in recent years, much interest has been shown towards un-stationary sensors (e.g., Robomote [9]), that can re-deploy themselves according to the needs of the application. These sensors are termed as mobile sensors and their networks as mobile sensor networks (MSNs).

WSN data, in form of online data streams, arrive at the base station as real-time updated data [10]. These online data streams are infinite, unbounded and have high continuous arrival rates which do not permit complete scanning of the entire data [11]. Various factors, such as limited power and transmission capabilities of sensors, hardware failures, power outages, and network issues like disruption, package collision and external noise, cause the transmitted data to fail to reach the base station and/or be corrupted. The sensors that 'generate' these missing data are called missing sensors. A major concern with any WSN is the issue of missing sensor data. Several approaches, such as ignoring missing data, using backup sensors, re-querying the network, and utilizing data estimating techniques to estimate the values of the missing data, have been proposed to address the issue of missing sensor data [15]. Ignoring missing data is not viable for sensitive applications; using backup sensors may lead to data duplication and is expensive; and re-querying the network is unrealistic in terms of time and resource efficiency. The approach that uses data estimation has shown to be the most promising solution; however, currently it is limited to SSNs only [15], [16], [17], [18]. To the best of our knowledge, no work has been proposed to estimate the values of the missing sensor data in MSN applications.

MSNs consist of sensors placed on mobile platforms like Robomote [9]. In addition to the issues common to any data stream application, MSN applications have certain additional constraints. MSN applications are broadly divided into relocation and continuous coverage based applications [7], [8]. The spatial relation between two sensors is distorted by the mobility of mobile sensors; hence the spatial relationship between two mobile sensors is difficult to obtain in MSNs. Moreover, the history data of a mobile sensor that are generated at different locations may not necessarily possess the spatial or temporal relationships with the data in the current round of sensor readings. Finally, mobile sensors have the capability of moving themselves which costs lots of energy; so power outage occurs more often on mobile sensors than on static sensors; hence, instances of missing data are more pronounced in MSNs.

In this paper we propose a data mining based solution for estimating the values of the missing sensor data in MSN applications, called DEMS (Data Estimation for Mobile Sensors). DEMS is a novel concept that addresses the issues associated with mobile sensors by utilizing virtual static sensors. DEMS establishes these virtual static sensors by dividing the entire monitoring area into hexagons and associating each hexagon's center with a virtual static sensor. It converts each mobile sensor reading into an equivalent virtual sensor reading. When a mobile sensor reading is missing, DEMS uses the spatial and temporal association rules among the virtual sensor readings that it discovers based on the history virtual sensor readings to compute the estimated value of the missing mobile sensor reading.

The rest of the paper is organized as follows: Section 2 discusses the related work; Section 3 describes DEMS; Section 4 presents the performance evaluation comparing DEMS with the three existing techniques: Average, Spirit [17], and TinyDB [13]; and Section 5 provides the conclusions and future work.

2. RELATED WORK AND ISSUES

Approaches for estimating the values of the missing sensor data (or approaches for estimating missing data for short), as of now, have been limited to SSNs only. TinyDB [13] is a prominent information extracting system for sensor networks. TinyDB does data estimation for a missing sensor by averaging the readings of other sensors for a particular round. However, it does not work well if a non-linear relationship exists among sensors and the sensors do not report similar readings. SPIRIT [17] uses auto-regression for finding correlations using hidden variables inside the history data of a sensor. It estimates missing data by predicting changes in data patterns using hidden variables as a summary of data correlation among all the history data. However, it does not consider the sensor readings from other sensors for the current round; therefore it is unable to find the current relationships among the data which may affect its accuracy. The Kalman filter [15] uses the dynamic linear model to predict missing data based on the history data. However, the dynamic nature of data distribution may introduce instances when the same sensor reports a completely different value in the current round compared to the previous rounds. This may cause erroneous results.

FARM [14] uses association rules among sensor readings to estimate missing data. It uses a novel data freshness framework to address the temporal nature of data. Further, it implements a data compaction scheme to store history data. Its estimated data are fairly accurate compared to those of statistical methods. However, its limitation is that it establishes association rules among similar sensor readings only; thus, only equivalent relationships are mined.

Mining Autonomously Spatio-Temporal Environmental Rules (MASTER) [16] is a comprehensive spatio-temporal association rules mining framework which provides both a competitive data estimation method and an exploratory tool to investigate the evolution of patterns of the sensor data in static sensor networks. MASTER is well equipped to discover spatial and temporal association rules among the sensors. This framework includes a novel data structure called MASTER-tree which stores the history data synopsis (the moments) for each sensor and represents the association rules among the sensors. An example of an association rule in MASTER is $S_1[10,20], S_2[40,90] \rightarrow S_3[30,40]$ where S_1, S_2 and S_3 are three sensors, S_1 and S_2 are called the antecedent sensors and S_3 is called the consequent sensor of the rule. This rule implies when the sensor reading of S_1 is between 10 and 20 and the sensor reading of S_2 is between 40 and 90, the sensor reading of S_3 would be between 30 and 40. Each node in the MASTER-tree represents a sensor except the root node which represents an empty node; and each path/sub-path starting from the root node represents an association rule. Hence a MASTER-tree is capable of representing any kind of relationships among the sensors which participate in the MASTER-tree.

MASTER limits the number of sensors in one MASTER-tree by clustering the sensors into small groups and producing an individual MASTER-tree for each cluster. The advantage of the clustering step is twofold: 1) the clustering step arranges spatially co-related sensors into a cluster, and 2) it limits the number of sensors in a MASTER-tree which restricts the exponentially large number of association rules into a more manageable number. As each data round arrives, MASTER finds the appropriate MASTER-tree for each sensor and updates the MASTER-tree based on the arrived sensor readings. At any particular time, if a sensor reading is missing, MASTER finds the appropriate MASTER-tree for the missing sensor and evaluates the support and confidence of the association rules where the missing sensor appears as consequent. MASTER finds the best association rule comparing the obtained support and confidence with the user-defined minimum support and minimum confidence. Finally, it uses the best association rule and the current sensor readings of the antecedent sensors in the best association rule to estimate the consequent sensor's reading. Interested readers are referred to [16] for further details.

MASTER was designed for SSNs. It has the following deficiencies. The cluster formation step is solely based on the spatial attributes of a sensor. In a MSN, the spatial data of a sensor are changing; therefore the prior knowledge about sensor locations is not enough for MSNs even though spatial clustering works very well in SSNs. One possible solution for this problem is re-clustering whenever a sensor changes its location, but re-clustering is very computation-intensive and may cause loss of the history data, and thus loss of history data synopsis (the moments) stored in the MASTER-tree. Hence location-based clustering for mobile sensors does not produce any meaningful result. Moreover, in a MSN, a reading of a sensor is accompanied by the location of the sensor. So, if a sensor is missing, it is very likely that the reading and the location from that sensor will be missing together. Hence the estimation technique must estimate both dimensions for the missing sensors, which means that location prediction has to be an inherent part of the technique.

In a SSN, association rule mining can be used to discover the relations among sensors. According to Tobler's first law of geography [22], geographically close sensors are more correlated than the distant one. In a MSN, the distance between the mobile sensors changes over time; therefore the correlation changes over time too. The association rules among the sensors represent the correlation among them. If the mobile sensors change their locations, the correlations among them change; hence the association rules previously obtained based on the sensor data will no longer be valid for the new locations. This

has two-fold implications on MASTER: 1) any previously explored rules may not be valid anymore; and 2) previously formed clusters may not be valid at all. In the extreme case, the history data from the same sensor may no longer be valid to estimate the missing data of the same sensor in the current round of data. This is because the old data are based on the previous locations of the sensor, whereas the new data are based on the new locations. So the methods (e.g., Kalman Filter [15]) which use history data to estimate new data will also become invalid in such a situation.

Motivated by the drawbacks of MASTER, in this paper we propose a new technique, called DEMS, for MSN applications. DEMS makes use of virtual static sensors that tackles the problems of location-aware clustering of real mobile sensors. It also tackles the problem of having no related history information for the current round of data from real mobile sensors. Moreover, DEMS addresses the issue of missing location of a real mobile sensor and is capable of predicting the next location for a missing real mobile sensor. The details of DEMS are presented in the next section.

3. THE PROPOSED DEMS

This section describes our technique, DEMS. It starts with a brief overview of DEMS followed by a detailed description of our novel concept of virtual static sensor and its significance. Finally it presents the MASTER-tree used for data mining and the estimation module for DEMS.

3.1 The Overview of DEMS

In DEMS, we exploit the spatial and temporal relations between sensor readings to estimate the missing sensor data. First we divide the entire monitoring area into hexagons based on a user-defined radius. Each hexagon corresponds to a virtual static sensor (VSS) placed at the center of the hexagon and covering the entire hexagon. A VSS is an artificial sensor, i.e. it does not exist physically in real life applications, but it exists in our technique as a synthetic sensor which mirrors a real static sensor. Each VSS has a unique identifier. DEMS converts the real mobile sensor readings into VSS readings based on the mobile sensors' current locations. Figure 1 shows *A* as the monitoring area covered by a MSN that is divided into 14 hexagons with 14 VSSs, V_1… $V_{14,}$ and 7 real mobile sensors, $M_{1..}$. $M_{7.}$

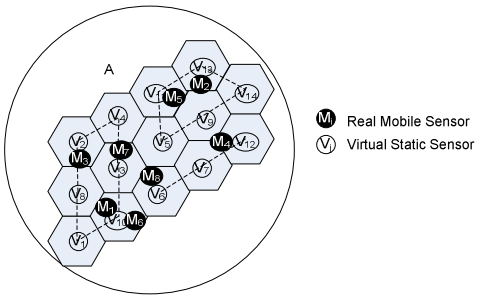


Figure 1. Monitoring area and hexagons

Using agglomerative clustering [23], DEMS clusters the VSSs based on their locations into clusters and creates a MASTER-tree for each cluster. The dotted lines that connect the centers of the hexagons in Figure 1 show three clusters (V_1, V_2, V_3, V_8, V_{10}), (V_6, V_7, V_{12}) and (V_5, V_9, V_{11}, V_{13}, V_{14}). MASTER-tree records the data for the VSSs. For each missing mobile sensor reading, its estimated value is computed using the three major steps: 1) mapping the missing real mobile sensor to its corresponding VSS; 2) estimating the missing VSS reading using the discovered spatial and temporal association rules among the history VSS readings, and 3) converting the estimated VSS reading into the corresponding real mobile sensor reading.

In a MSN, a sensor reading reported is accompanied by the sensor location where the reading was obtained. Whenever a mobile sensor reading is missing (we call this a missing mobile sensor for short), it is likely that both the location and the reading will be missing together. To find the appropriate location of a missing mobile sensor we always keep track of mobile sensors' locations. A mobile sensor's location is mapped to a hexagon and the consecutive locations of a mobile sensor are mapped to a sequence of hexagons. We refer to a sequence of hexagons as a mobile sensor's trajectory. We mine the mobile sensor trajectories and predict the missing location based on the history trajectories. Morzy [20] proposed a pattern tree based approach for mining trajectories and predicting future locations, which we adopt for DEMS. DEMS maintains a single pattern tree of trajectories for all the mobile sensors. As small devices like sensors often use the same protocol for relocation [7], [9], it is reasonable to assume that they have similar patterns of movement; therefore DEMS maintains a single pattern tree of trajectories for all the mobile sensors and uses a single pattern tree instead of an individual pattern tree for each mobile sensor. This trajectory pattern tree is used to predict a missing mobile sensor's location. The predicted location is used to map a mobile sensor to a VSS. Since sensors repeat the mobility pattern for relocation, history based trajectory mining is more promising than random walk models.

3.2 The Virtual Static Sensor

In SSNs, every sensor monitors a fixed region and a sensor's reading reflects an event occurring within this region; but in MSNs, owing to their mobile nature, the region being monitored varies with time. However, as in SSNs, the sensor readings for MSNs still reflect events occurring within a particular region. Our concept of virtual static sensors is directly motivated by the above fact. Every VSS, like sensors in SSNs, 'monitors' a fixed region called its coverage area. An event occurring within a VSS's coverage area is reflected in its readings. However, unlike sensors in SSNs, VSSs do not have real existence and do not 'report' data to a base station. On the contrary, they are 'created' in our technique virtually to ease the spatio-temporal data mining.

A VSS reports a reading if there exists at least one real mobile sensor in the coverage area. A VSS is *active* if it reports in the current round and is *inactive* otherwise. VSS readings are readings of the real mobile sensor(s) which are present in the VSS's coverage area. In situations when multiple real mobile sensors are in a VSS's coverage area, the VSS reports the average of all the real mobile sensors' readings. There are two reasons for considering the average reading: 1) multiple sensors monitoring the same small coverage area most likely will report similar readings; and 2) any event occurring in the common coverage area will be reflected in the readings of all the sensors monitoring that area. As a hexagon is the atomic coverage region in DEMS, the radius of each hexagon is usually small enough to assure the variance of real sensors' readings from the

same hexagon to be minimal, and averaging all readings from sensors from the same hexagon will be close to the real value of the corresponding region. A VSS is called a *missing VSS* if one real mobile sensor exists or expected to exist within the coverage area of that particular VSS and the reading from the real mobile sensor is missing.

The total monitoring region for any MSN or SSN is fixed either due to application specifications or hardware constraints. However, we further sub-divide the MSN's monitoring region into fixed size hexagons with a VSS 'covering' each particular hexagon. We choose hexagonal coverage area as they do not suffer from overlapping or uncovered regions as in the case of circular coverage area. Thus, in our monitoring area, we do not encounter regions where a real mobile sensor can map to multiple VSS (for overlapping regions) or cannot map to any VSS (for uncovered regions). Two virtual static sensors are neighbors if their covered hexagons share at least one edge. Due to the static nature of VSSs, they have a static spatial relation among themselves and can be co-related too. Finally consecutive readings from a VSS are originated from the same location and can show temporal relationships among them.

```
Procedure mapReal2Virtual(RealSensorData listRSData, VirtualSensorData
listVSData)
 1   for each real sensor rs
 2     if(rs is not missing)
 3       location ← listRSData(rs).Location
 4       vs ← findVirtualSensor(location)
 5       listVSData(vs).addReading(listRSData(rs).Reading)
 6     else
 7       location ← predictLocation(rs)
 8       vs ← findVirtualSensor(location)
 9       listVSData(vs).status←missing
10   end loop
11   for each virtual static sensor vs
12     if(listVSData(vs) has data)
13       listVSData(vs).status←active
14   listVSData(vs).reading←average(listVSData(vs).Readings)
15     else
16       if(listVSData(vs).status is not missing)
17         listVSData(vs).status ←inactive
18   end loop
end procedure
```

Figure 2. Mapping mobile sensor readings to virtual static sensor readings

Hence VSS readings are directly stored in our MASTER-tree. So, in DEMS, the MASTER-tree represents the relationships among the VSSs. We assume that at any instance, all the mobile sensors report their readings to the base station, which is then mapped to the corresponding VSSs. Figure 2 shows the mapping algorithm in details. For each real mobile sensor, DEMS finds the appropriate VSS (lines 3 & 4) using a geometric mapping between location and hexagon. If the location of the real mobile sensor is missing, DEMS predicts the expected location for the real mobile sensor and maps it to the appropriate VSS for that predicted location. If the mobile sensor reading is missing, DEMS marks the corresponding VSS as missing. Finally, in the loop from lines 11 to 18, each VSS is marked appropriately as active, inactive or missing. At any particular time, only the active virtual static sensors are stored in their appropriate MASTER-trees.

3.3 The MASTER-tree Projection Module

A MASTER-tree is like a pattern tree, which is used to represent arbitrary relationships among all Boolean itemsets [19]. A pattern tree is equivalent to a spanning tree of a binary hypercube which represents all possible Boolean items relationships; but the computational complexity of a pattern tree is exponential. However, grouping items into a set of clusters and pruning the pattern tree or its equivalent hypercube lowers the computational complexity. A pattern tree unduly favors only the right most leaf node and extracts the relationships of this node with all other nodes. A MASTER-tree does not suffer from those issues of a pattern tree. It combines the various pattern trees regarding each node and prunes the common paths in the resulting tree and forms a new tree called a MASTER-tree [16].

In a MASTER-tree, each tree node represents a VSS. The data distribution of a particular VSS node over a particular vector space is stored in each node. The complete vector space, in which the VSS readings occur, is discretized into a finite number of cells. Technically, for each cell, an arbitrarily accurate data distribution function or probability distribution function can be represented by an infinite number of moments in statistical theory. However, computationally, only a finite number of moments plus element counters are stored in the MASTER-tree nodes (typically the first four moments). An element counter is the number of VSS readings belonging to the cell associated with the corresponding MASTER-tree node. For each cell, a few moments are stored, and the cells across nodes are linked following the MASTER-tree paths. These cells and links form a grid structure (GS). As GS depends on a finite number of cells and a fixed number of nodes in a particular cluster, it does not grow exponentially with the increase in the number of rounds of sensor readings. Thus, the MASTER-tree projection module is to establish a MASTER-tree for each cluster and then to incrementally update the GS as a new round of sensor readings arrives. This maintains the up-to-date association rules among the VSSs in a cluster to serve data analysis purposes. Interested readers are refer to [16] for details about this module.

3.4 The Data Estimation Module

The data estimation module computes the estimated value for the missing mobile sensor. Initially, the location of the missing mobile sensor is predicted based on the user-defined minimum support and minimum confidence using Morzy's approach [20]. If the algorithm fails to predict the next location, DEMS uses the last reported location of the missing mobile sensor as its current location. Location prediction is preceded by mapping the missing mobile sensor to the corresponding VSS, which is called missing VSS. The estimated missing mobile sensor reading is the estimated missing VSS reading computed from the MASTER-tree.

The data estimation module accomplishes the task in an iterative way. First it obtains the prior distribution of the missing VSS (mVSS) from the MASTER-tree, i.e., the rule ø → mVSS (here ø means empty). If the rule satisfies the user-defined error margin and the minimum support and minimum confidence thresholds, the rule holds and the estimated value is produced by taking the average of the prior distribution of mVSS. However, if the error margin requirement is not satisfied, the related information from the other tree nodes (VSSs) is considered for re-estimation. Here, the data estimation module chooses one more new antecedent node to infer the mVSS's reading. As every node represents a VSS, a node can be an antecedent node if the corresponding VSS is active. The initial relevant subspace for the antecedent node is simply the cell picked up based on its current reading. When the actual support does not satisfy the

minimum support threshold, the relevant subspace is augmented iteratively until the actual support is no less than the minimum support. However, if the support requirement cannot be satisfied even if the relevant space reaches its upper limit, i.e., the complete subspace, the module removes this node and considers a new prior node. This process of adding a new antecedent node is repeated until the estimation procedure meets one of the following conditions: 1) a rule that satisfies the minimum support, minimum confidence and maximum error margin is found, or 2) no more nodes within the cluster is to be added to the antecedent nodes set. The procedure then returns the estimated value using the last expected value (the average) over the obtained consequent subspace. The estimated mVSS's reading is directly used as the estimated reading for the missing mobile sensor.

4. EXPERIMENTAL DESIGN AND RESULTS ANALYSIS

In this section, we compare DEMS with two existing algorithms: SPIRIT [17] and TinyDB [13]. Although both TinyDB and SPIRIT are designed for static sensors, it can be argued that they can still be used for data estimation for mobile sensors disregarding sensor's mobility. We also compare it with the Average which is a statistical baseline method where the missing reading is estimated by averaging all other known sensor readings of the current round.

4.1 Experimental Datasets

4.1.1 The DAPPLE project dataset

The real life dataset is obtained from the DAPPLE project [21]. The data are about carbon monoxide (CO) readings collected over a period of two weeks around Marylebone Road in London. The mobile sensors monitoring the atmospheric CO level are attached to PDAs which store these readings. The data sampling rate of the sensors is once every second. The software on the PDAs generates log files containing the CO levels with the locations and the timestamps associated with the readings. Each reading was carried out with a single sensor kit every second for a duration of about 45 minutes over a two weeks period. Simultaneous use of multiple sensors (usually three) was limited to some days only. For our experimental purposes, we considered the instances when three sensors were simultaneously recording CO pollution levels for a considerable period of time. We chose Thursday, 20th May 2004, when three sensors were simultaneously recording for about 32 minutes, resulting in 600 rounds (after disregarding the missing rounds) of CO readings.

4.1.2 The Factory Floor Temperature Dataset

Besides the above real life application dataset, we also synthesized a factory floor temperature dataset [12] which exhibits dynamically changing phenomena. In this experiment, machines are placed on a grid floor. Initially, all machines are off and the starting temperatures for all grid points are set to zero. The boundary grid point temperature is held at zero throughout the experiment. Then some machines will be turned on for a number of rounds; the temperatures on these machines will reach a high constant temperature and heat will disperse on the floor. At each time step, a grid point is updated using the heat transfer formula used in [12]. In this simulation, 100 mobile sensors were roaming around in random directions to monitor the factory floor and reported the temperature readings from different locations at different points in time. In our simulations, we sampled the mobile sensor readings once per hour. In total we gathered 5000 rounds of readings from 100 sensors.

4.2 Performance Comparison Studies

In this section we compare the performances of DEMS, Average, SPIRIT [17], and TinyDB [13] in terms of mean absolute error (MAE). MAE is calculated using the following formula: $MAE = \frac{\sum_{i=1}^{n}|e_i - v_i|}{n}$ where e_i is the estimated value, v_i is the original value for the i-th data point, and n is the total number of data points. MAE is thus the magnitude, not the percentage, of the error. We specifically study the impacts of the number of rounds of sensor readings on the estimation accuracy.

4.2.1 Results for the DAPPLE Project Dataset

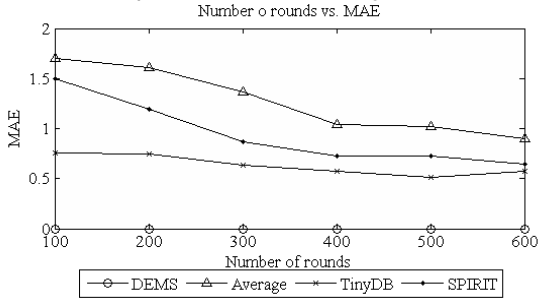


Fig 3. Number of rounds vs. MAE for the DAPPLE project dataset

Figure 3 shows the change of MAE with the change of number of rounds of sensor readings. The MAE value of 0 for DEMS implies that DEMS estimates the missing data with no error. A possible reason is that the DAPPLE project dataset has very few variations (the CO levels are within the range 0~6) and the sensors have very high spatial correlations. In most cases the readings in the same hexagon are the same. Hence, DEMS produces a zero error in terms of MAE. The MAEs for other approaches are comparatively high at the beginning and become stable at the end as the number of rounds increases.

Table 1. Average MAEs for the DAPPLE project dataset

Approach	Average MAE
DEMS	0
Average	1.2717
TinyDB	0.6331
SPIRIT	0.9437

Table 1 shows the average MAE for all the approaches. DEMS almost perfectly estimates the missing values while Average gives the highest error compared to SPIRIT and TinyDB.

4.2.2 Results for the Factory Floor Temperature Dataset

Table 2. Average MAEs for the factory floor temperature dataset

Approach	Average MAE
DEMS	2.2538
Average	14.7787
TinyDB	6.9621
SPIRIT	4.7472

We performed a similar study for the factory floor temperature dataset. This dataset have more variations (temperatures are in

the range 0~100C) compared to the DAPPLE project dataset. Figure 4 shows the change of MAE with respect to the change of number of rounds. The MAE for each approach remains almost constant when the number of rounds changes. As this dataset has more variations than the DAPPLE project dataset, even though DEMS still performs better than the other techniques, its performance is not as good as its performance with the DAPPLE project dataset.

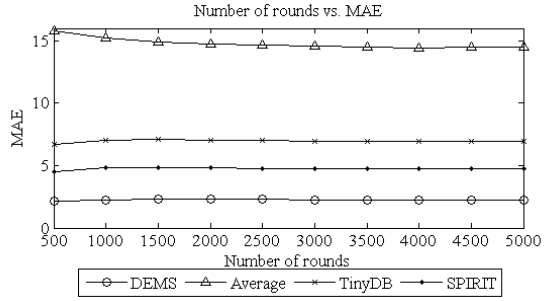


Fig 4. Number of rounds vs. MAE for factory floor temperature dataset

Table 2 shows the average MAE for all the approaches. The average errors produced by Average, SPIRIT and TinyDB are about seven times, three times, and two times more than that produced by DEMS, respectively. DEMS is thus very effective in estimating missing sensor data.

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new technique (DEMS) to estimate missing data in MSN applications. Experimental results show that the estimated values computed by DEMS are more accurate than those produced by the existing techniques: Average, SPIRIT [17], and TinyDB [13]. For future work, we will consider the case when multiple mobile sensors report data at different times. We envision scenarios where considerable delays may exist between each sensor's readings. Finally as DEMS currently is designed for single hop MSNs only, we plan to expand the scope of DEMS to include multi-hop MSNs, mobile base station, and clustered MSNs.

6. ACKNOWLEDGMENTS

This work has been supported in part by the NASA under the grant No. NNG05GA30G.

7. REFERENCES

[1] T. Haenselmann; An FDL'ed Textbook on Sensor Networks, GNU Free Documentation License, 2005.

[2] A. Mainwaring, D. Culler, J. Polastre , R. Szewczyk, J. Anderson; Wireless sensor networks for habitat monitoring, 1st ACM international workshop on Wireless sensor networks and applications, 2002.

[3] Metar. http;//metar.noaa.gov/, Last Accessed - January 2010.

[4] L. Schwiebert, S.Gupta, and J.Weinmann; Research challenges in wireless networks of biomedical sensor, MobiCom, 2001.

[5] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan and K.K Saluja; Sensor Deployment Strategy for Target Detection, 1st ACM International Workshop on Wireless Sensor Networks and Applications, 2002.

[6] S. Meguerdichian, F. Koushanfar, M. Potkonjak and M.B.Srivastava; Coverage Problems in Wireless Ad-Hoc Sensor Networks, INFOCOM, 2001.

[7] B. Liu, Peter Brass, Olivier Dousse, Philippe Nain, and Don Towsley; Mobility Improves Coverage of Sensor Networks, MobiHoc, 2005.

[8] G. Wang, G. Cao, T. parta, and W. Zhang; Sensor Relocation in Mobile Sensor Networks, INFOCOM, 2005.

[9] G.T. Sibley, M.H. Rahimi and G.S. Sukhatme; Robomote – A tiny Mobile Robot Platform for Large-Scale Sensor Networks, ICRA, 2002.

[10] N. Jiang , Le Gruenwald; Research issues in data stream association rule mining, SIGMOD Record 2006.

[11] S. Guha, N. Koudas, K. Shim; Data Streams and Histrograms, ACM Symposium on Theory of Computing, 2001.

[12] A. Silberstein, K. Munagala, and J. Yang; Energy-Efficient Monitoring of Extreme Values in Sensor Networks, ACM SIGMOD, 2006.

[13] S. Madden, M. Franklin, J. Hellerstein and W. Hong; TinyDB: An Acquisitional Query Processing System for Sensor Networks, Transactions on Database Systems, 2005.

[14] L. Gruenwald, H. Chook, M. Aboukhamis; Using Data Mining to Estimate Missing Sensor Data, ICDMW, 2007.

[15] N. Vijayakumar and B. Plale; Missing Event Prediction in Sensor Data Streams Using Kalman Filters, book chapter in Knowledge Discovery from Sensor Data, Taylor and Francis/CRC Press, 2009.

[16] H. Chok and L. Gruenwald; An online spatio-temporal association rule mining framework for analyzing and estimating sensor data. IDEAS, 2009.

[17] S. Papadimitriou, J. Sun, and C. Faloutsos; Streaming Pattern Discovery in Multiple Time-series, VLDB, 2005.

[18] L. Gruenwald, H. Yang, S. Sadik, R. Shukla; Using Data Mining to Handle Missing Data in Multi-Hop Sensor Network Applications, MobiDE, 2010.

[19] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu.in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.); Mining Frequent Patterns in Data Streams at Multiple Time Granularities. Next Generation Data Mining, AAAI/MIT, 2003.

[20] M. Morzy; Mining Frequent Trajectories of Moving Objects for Location Prediction, Machine Learning and Data Mining in Pattern Recognition, LNCS, 2007.

[21] UCL Carbon Monoxide Data Collection at Dapple Site, http://www.cs.ucl.ac.uk/research/vr/Projects/envesci/DAPPLE2004/UCLDAPPLE.html, Accessed May 2010.

[22] W.Tobler; A Computer Movie Simulating Urban Growth in the Detroit Region, Economic Geography, 1970.

[23] W. Day and H. Edelsbrunner; Efficient Algorithms for Agglomerative Hierarchical Clustering Methods, Journal of Classification, 1984.

PAO: Power-Efficient Attribution of Outliers in Wireless Sensor Networks

Nikos Giatrakos
Dept. of Informatics
University of Piraeus
Piraeus,Greece
ngiatrak@unipi.gr

Yannis Kotidis
Dept. of Informatics
Athens University of
Economics and Business
Athens,Greece
kotidis@aueb.gr

Antonios Deligiannakis
Dept. of Electronic and
Computer Engineering
Technical University of Crete
Crete,Greece
adeli@softnet.tuc.gr

ABSTRACT

Sensor nodes constitute inexpensive, disposable devices that are often scattered in harsh environments of interest so as to collect and communicate desired measurements of monitored quantities. Due to the commodity hardware used in the construction of sensor nodes, the readings of sensors are frequently tainted with outliers. Given the presence of outliers, decision making in sensor networks becomes much harder. In this work, we introduce PAO, a framework that can reliably and efficiently detect outliers in wireless sensor networks. PAO significantly reduces the bandwidth consumption during the outlier detection procedure, while being able to operate over multiple window types. Moreover, our framework possesses the ability to operate in either an exact mode, or an approximate one that further reduces the communication cost, thus covering a wide variety of application requirements.

1. INTRODUCTION

Many monitoring applications rely on wireless sensory infrastructures in order to obtain measurements of the surrounding environment. Examples include habitat monitoring applications that collect meteorological data (like temperature, pressure, humidity etc), military surveillance applications that track movement of personnel or detect potentially hazardous chemicals, as well as vehicle tracking and traffic surveillance applications. Despite their diversity and differences, all such applications share the need to collect measurements that accurately reflect the conditions of the physical world being monitored. However, sensory infrastructures, in order to provide an economically viable solution, typically rely on inexpensive hardware used for the construction of the nodes. As a result, sensor nodes often generate imprecise individual readings due to interference or failures [10]. In several application scenarios, sensor nodes are often thrown in hazardous environments and need to operate in an unattended manner for long periods of time. Such nodes, may be exposed to severe conditions that adversely affect their sensing elements, thus yielding readings of low quality. As an example, the humidity sensors on the popular MICA mote is very sensitive to rain drops [5]. A question that naturally arises is whether (and how) one can build applications that take mission-critical decisions, given the lack of trust on the baseline measurements provided as input by the sensory infrastructure.

DMSN '10, September 13, 2010, Singapore

In order to address this question, a flurry of recent work has targeted the problem of outlier detection in sensor networks [3, 5, 12, 17, 18]. Although the detection of outliers is an old research problem, the particular challenges introduced when considering ad-hoc sensor networks render conventional outlier detection algorithms [2] unsuitable for this new setting. In particular, sensor nodes often have limited processing and storage capabilities. As a result, sensory data that is continuously collected by the nodes may be maintained in memory for only a limited amount of time. Moreover, since data is collected continuously (typically in predefined intervals, called epochs), outlier detection needs on-line mechanisms that will work in this restrictive, streaming setting. However, the main constraint in sensor network applications is often the limited energy capacity of each sensor node. In many applications, sensor nodes are powered by batteries that cannot be easily replaced if depleted, given that nodes are often thrown in remote sites. Therefore, in order to ensure the longevity of the sensor network, we need to devise techniques that can detect outliers in an energy-efficient manner, thus reducing the energy drain of the nodes. It is well understood that radio communication is by far the the biggest culprit in energy drain [13]. This means that a central collection of all sensory data (and subsequently, computation of outliers using existing centralized techniques) is not feasible since it results in high energy drain, due to the large amounts of transmitted data. Hence, what is required are continuous, distributed and in-network approaches that reduce the communication cost and manage to prolong the network lifetime.

In this paper we introduce PAO (PAO stands for Power-efficient Attribution of Outliers), an outlier detection framework tailored for the particular constraints faced in typical sensor networks applications. Our framework follows the in-network paradigm, meaning that computation of outliers is performed inside the network, avoiding in this way the communication of raw sensor measurements to the base station. Furthermore, PAO possesses the ability to perform over multiple types of windows of observations collected by motes. Similar to previous techniques [5, 8], our framework takes into account both temporal and spatial correlations in order to characterize the readings of a sensor node. Temporal correlations are captured by considering the latest measurements of a sensor node and by computing localized regression models out of them. These compact models are of fixed size and are used to replace the original data values, reducing in this way the size of data that is transmitted in the network. In PAO, we adopt a clustered network organization [14, 19], where nodes communicate their regression models to a clusterhead, which computes the similarity amongst the latest values of any pair of sensors within its cluster. Based on the performed similarity tests and a desired minimum support specified by the posed query, each clusterhead generates a list of *potential* outliers within its cluster. This list is then communicated, in a second (inter-cluster) communication phase of PAO, among the clus-

terheads, in order to allow potential outliers to gain support from measurements of nodes that lie in other clusters. The whole process is sketched in Figure 2.

In order to alleviate clusterheads from comparing models from all nodes within their cluster, we introduce PAO+, an extended version of the original framework where additional nodes within each cluster are utilized for that purpose. This extended scheme is made possible by introducing a simple, yet effective hashing scheme over the regression parameters computed by the sensor nodes. The benefits of this extended scheme are twofold. Not only do we spread the load into multiple nodes, but also (as will be explained) manage to avoid many comparisons between regression models of motes that are provably not similar and, thus, cannot support each other. The PAO+ scheme also offers a load balancing mechanism that periodically adopts the hashing functions to the characteristics of the collected data, resulting in more effective comparison pruning.
The contributions of this paper can be summarized as follows:

1. We introduce PAO, an in-network outlier detection framework that permits computation of outliers in a clustered sensor network. PAO is capable of performing over different window types, it takes into account both temporal and spatial correlations among the measurements of the nodes and utilizes simple regression models in order to reduce communication overhead. We provide techniques for suppressing update messages by the motes, in continuous queries, resulting in fewer transmitted bits.
2. We describe PAO+, which extends PAO with a novel load balancing and comparison pruning mechanism. The proposed extensions alleviate clusterheads from excessive processing and communication load and result in a more uniform, intra-cluster power consumption and prolonged network unhindered operation.
3. We present a detailed experimental analysis of our techniques using real data sets. Our analysis demonstrates that our techniques manage to detect outliers using only a fraction of the bandwidth that a centralized approach would require.

This paper proceeds as follows. In Section 2 we comment on related work. Section 3 presents preliminary concepts. Our PAO framework is introduced in Section 4, while its extensions are discussed in Section 5. Section 6 presents our experimental evaluation, while Section 7 includes concluding remarks.

2. RELATED WORK

Recently, substantial effort has been devoted to the development of efficient outlier detection techniques that manage to pinpoint motes producing low quality readings or observe interesting undergoing phenomena so that proper actions can be taken [21]. [6, 10] introduce data cleaning mechanisms over sensor data streams after their central collection at the query source. Nevertheless, the central collection of data is not feasible nor desired as it has a cumulative effect on the amount of communicated data, which in turn depletes the residual energy of the motes.

Localized voting protocols [3, 18] have been proposed to determine faulty motes in completely ad-hoc network formations. However, such voting schemes are prone to errors when motes generating imprecise measurements are not able to communicate with each other due to physical obstacles or other unpredictable disturbances in their surrounding [5]. A fused weighted average scheme is proposed in [9] where a fuzzy mechanism is utilized to infer the correlation among sensor readings. In other related work, [22] makes use of a weighted moving average to clean imprecise samples while a histogram-based outlier attribution method is presented in [16].

The authors of [17] foster kernel functions to estimate the data distribution of motes and subsequently detect distance-based outliers leveraging this information. The work of [5] manages to provide outlier reports on par with the execution of aggregate and

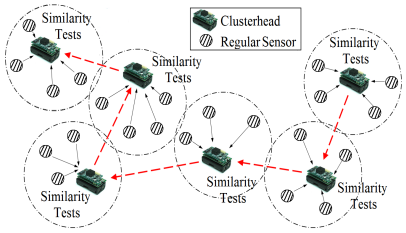


Figure 1: Main Stages of PAO. Step 1: Intra-cluster communication between regular motes and clusterheads (solid black arrows). Step 2: Similarity tests are performed by clusterheads. Step 3: An approximate TSP problem is solved, potential outliers are exchanged (dashed red arrows). The final outlier list is transmitted to the base station (not shown).

group-by queries posed to an aggregation tree [1, 13, 20]. It thus excludes extraordinary measurements avoiding the distortion of the final aggregate result and simultaneously allows users to acquire important information of motes exhibiting abnormal behavior. The recent work of TACO [8], manages to efficiently determine outliers by providing a mechanism based on Locality Sensitive Hashing [7], which trades bandwidth consumption for accuracy during the outlier detection procedure in a straightforward way. However, PAO is applicable to multiple types of window queries. Message suppression schemes in sensor networks for continuous aggregate queries have been studied in [4, 15]. Our work differs in that we do not suppress raw measurements but updates to model parameters instead.

3. PRELIMINARIES

3.1 Network Model

We adopt an underlying network structure where motes are organized into clusters (shown as dotted circles in Figure 2) using any existing network clustering algorithm [14, 19]. Queries are propagated by the base station to the clusterheads, which, in turn, disseminate these queries to sensors within their cluster.

3.2 Analyzing Trends in Mote Time Series

A time series constitutes a sequence of observations $Y_{t_0}, Y_{t_1}, \ldots, Y_{t_{w-1}}$, where $t_0 < t_1 < \cdots < t_{w-1}$, regarding a studied attribute of interest Y_t, in w different time instances. In our sensor network setting, a posed outlier detection query dictates the epoch parameter e, which is the time interval between consecutive samples. As a result, after obtaining w quantities every mote S_u has formed a series $Y_t^{S_u}$ with $t = (0, e, \ldots, (w-1) \cdot e)$ as the vector of the corresponding timestamps.

Time series analysis techniques aim at capturing the implied behavioral pattern in the observed data. A fundamental component describing the existing patterns is the *trend* of the series, which is able to describe the rate of change on the values of an attribute. This in turn provides a compact picture of the presence of interesting phenomena imprinted on previously acquired samples. A simple yet popular way to depict the trend of time series data is through linear models in which:

$$\hat{Y}_t = \hat{a} \cdot t + \hat{b} \tag{1}$$

According to Equation 1, the value of a studied attribute given the time vector is expected to be encompassed by a line with parameters $\hat{a}$ and $\hat{b}$ taking values:

$$\begin{cases} \hat{a} = \frac{12(\sum_{i=0}^{w-1} t_i \cdot Y_{t_i} - \frac{e \cdot w \cdot (w-2)}{2(w-1)} \sum_{i=0}^{w-1} Y_{t_i})}{e^2 \cdot w \cdot (w^2+2)} \\ \hat{b} = \overline{Y}_t - \hat{a} \cdot \frac{e \cdot w}{2} \end{cases} \tag{2}$$

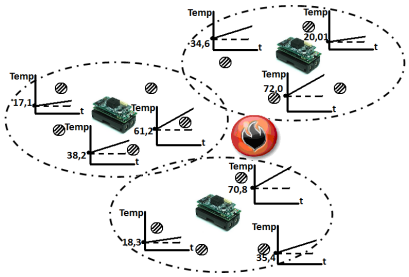


Figure 2: Trends and intercept points in mote time series depending on the spatial proximity to the source of the fire burst.

$\overline{Y}_t$ refers to the mean of Y_{t_i}s. Parameter $\hat{a}$ expresses the *slope* of the linear model, while $\hat{b}$ represents the *intercept* point between the line and Y_t axis. Hence, $\arctan(\hat{a})$ computes the actual angle $\measuredangle Y_t$ that the linear model's slope forms in respect with the time axis and $-\frac{\pi}{2} < \measuredangle Y_t < \frac{\pi}{2}$.

As an example, consider a sensor network in a forest designed to sample attributes such as temperature, humidity etc. Should a fire burst arise (Figure 2) nearby motes will collect increasing temperature values. The absolute sampled values actually depend on the radius around the source of the event that a mote is placed but its rate and, thus, the trend of the corresponding time series will be similar. In other words, upon utilizing a linear representation so as to model the trend on motes data, parameter $\hat{b}$ regards the proximity of a mote to the source of an event, contrary to $\hat{a}$ which shows the actual rate in change of values. The same observation holds for other physical attributes such as humidity (i.e., flood occurrence where motes obtain increased humidity values sensed in the air), sound vibrations, radiance measurements etc.

Nevertheless, in practice samples within a specific time window may exhibit extraordinary deviation where no clear trend seems to appear. Situations like these should be handled differently due to the lack of a certain behavioral pattern. The question is how could someone check whether such a pattern does exist. To achieve that we reside to the *correlation of determination*, which shows the amount of variance in Y_t explained by the model:

$$R^2 = \frac{\sum_{i=0}^{w-1}(\hat{Y}_{t_i} - \overline{Y}_t)^2}{\sum_{i=0}^{w-1}(Y_{t_i} - \overline{Y}_t)^2}, 0 \leq R^2 \leq 1 \quad (3)$$

High values of R^2 validate that a trend exists and is well modeled by Equation 1. On the contrary, low values indicate the absence of this kind of motive.

3.3 Outlier Definition

Based on our previous discussion we formalize our definition of outlying values. We assume that the posed outlier detection query has specified a couple of parameters p, $\measuredangle_{thres}$, whose meaning will be introduced shortly. Given the time series $Y_t^{S_u}, Y_t^{S_v}$ of motes S_u, S_v we initially utilize Equation 3 to check whether behavioral patterns that can be described by linear models occur based on threshold p. That is, we simply check whether $R^2 \geq p$ for $Y_t^{S_u}$ and $Y_t^{S_v}$, respectively. Please note that each test can be performed independently by the corresponding mote. If this is true, then, as already mentioned, we only need to compare the trends based on the value of $\hat{a}$s and more precisely on the equivalent angles $\measuredangle Y_t^{S_u}, \measuredangle Y_t^{S_v}$.

Given a similarity threshold $\measuredangle_{thres}$ specified by the posed query we consider $Y_t^{S_u}, Y_t^{S_v}$ as similar if:

$$|\measuredangle Y_t^{S_u} - \measuredangle Y_t^{S_v}| \leq \measuredangle_{thres} \quad (4)$$

Acquired samples that do not pass the R^2 linearity test do not exhibit any profound motif and should be compared separately. Such cases can be handled using the *cosine coefficient* so as to compute the angle similarity [8] between vectors Vec_{S_u}, Vec_{S_v}. Value vector $Vec_{S_u} \in \mathbb{R}^w$ contains the measurements of S_u during the latest w samples and the angle similarity in this case is calculated by $\measuredangle(Vec_{S_u}, Vec_{S_v}) = \arccos \frac{Vec_{S_u} \cdot Vec_{S_v}}{||Vec_{S_u}|| \cdot ||Vec_{S_v}||}$.

As in [5, 8], we require our techniques to be resilient to environments where spurious readings originate from multiple node time series, due to a multitude of different and unpredictable factors. Thus, for a mote not to be classified as an outlier it should be found similar with at least *minSup* other motes. The value of *minSup* can be expressed either as an absolute value or as a percentage of motes.

4. OUR PAO ALGORITHM

We now present our PAO algorithm in detail. We assume that an outlier attribution query of the form:

SELECT $c.S_u$
FROM Clusterheads c
WHERE $c.Support_{S_u} < minSup$
USING [
SAMPLING PARAMETERS ($Interval = e$,$Size = w$),
TESTS(Linearity p, Similarity $\measuredangle_{thres}$),
CHECKS ON <set of specifications on similarity tests>),
WINDOW TYPE={ Disjoint | Sliding } with ϵ]

has been posed to the sensor network. The parameters of the query have been presented in the previous sections. Regarding the set of specifications noted in CHECKS ON line of the USING compartment, we note that it refers to motes that may find support outside their cluster based on a set of static specifications. For instance, users may allow motes within a certain radius to be able to witness each other, irrespectively of whether they have been assigned to the same cluster, as they are expected to be able to sense similar conditions (i.e, the fire burst in the example of Figure 2). The last line of the query refers to the window type (disjoint or sliding). In a nutshell, using disjoint windows the query evaluation utilizes a set of w new samples (not used in previous query evaluations - this is often referred to as a tumble), while sliding windows always utilize the latest w observations (thus at each step taking into account $w-1$ observations that were also used in the previous evaluation, but then also adding the latest observation by the mote). Parameter ϵ that accompanies the window type involves a message suppression choice provided by PAO so as to support the potential for approximate detection of outliers with further reduced communication costs, as it will be explained at the end of the current section.

PAO at Individual Motes. After it receives a corresponding query, every mote S_u in the network assembles a time series $Y_t^{S_u}$. Initially, S_u computes $\hat{a}, \hat{b}$ using Equation 2, the correlation of determination R^2 using Equation 3 and performs the linear trend existence test by checking whether $R^2 \geq p$. Recall that p expresses a tolerance on the deviation of the collected measurements. In practice, an amount of this deviation is due to systematic calibration errors of the inexpensive hardware used in the construction of sensor nodes. As a result, knowing the specifications of the available mote hardware infrastructure, users can appropriately set the desired value for p. Subsequently, depending on the result of the latter test, S_u calculates $\measuredangle Y_t^{S_u} = arccos(\hat{a})$ which is then transmitted to the clusterhead. $R^2 < p$ results in communicating the analytical form Vec_{S_u} of samples to the clusterhead.

Intra-cluster Processing. Clusterheads receive data from motes in their cluster and organize this information in a tabular format with columns S_u, $\measuredangle Y_t^{S_u}$ or Vec_{S_u} for motes that did not pass the linearity test, and $Support_{S_u}$.

Data collection is horizontally fragmented between the clusterheads and further separated in motes with captured behavioral pat-

terns and those which do not adapt to the model. The $Support_{S_u}$ column is set to 0 at the beginning of this phase. Subsequently, each clusterhead performs similarity tests as in Equation 4 on the first category of motes, while applying $\measuredangle(Vec_{S_u}, Vec_{S_v})$-based tests for the second. Each successful test increases the support of the participating motes by 1. At the end of the procedure, each clusterhead forms a list of tuples $\langle S_u, \measuredangle Y_t^{Su}, Support_{S_u}\rangle$ and $\langle S_u, Vec_{S_u}, Support_{S_u}\rangle$ for sensors that did not manage to obtain enough witnesses to reach *minSup*.

Inter-cluster Processing. As already noted, lists of motes with $Support_{S_u} < minSup$ are not final outliers since the query may have allowed motes in different clusters to be tested for similarity. Motes in the lists extracted by cluster C_i that are not subjected to such kind of specifications can be directly reported to the query source. Otherwise, triplets are placed in a list $PotOut_{C_i}$ of potential (i.e., not yet determined) outliers. Given the current cluster as the starting node, query-specified clusterheads as intermediate sites and the base station for the destination, the intercluster communication problem is modeled as a TSP according to which $PotOut_{C_i}$s are exchanged between clusterheads participating in the path. The TSP problem can be solved by the base station after clusterhead election. Every $S_u \in PotOut_{C_i}$ that manages to reach *minSup* is excluded from the list that will be forwarded to the next site.

Approximate Processing over Multiple Window Types. So far, the procedure presented in PAO reduces the communication burden only for disjoint time windows. In other words, motes collect w quantities, form corresponding time series, and intra- as well as inter-clustering processes are then triggered. Provided that S_u succeeds in its linearity test, only $\measuredangle Y_t^{S_u}$s (instead of the entire series) are transmitted. Subsequently, these steps are repeated every w new measurements. On sliding window queries, new results are to be provided based on the $w-1$ previous observations and the latest w-th measurement, obtained every e time units. In this case, letting motes transmit $\measuredangle Y_t^{S_u}$ does not provide any savings in communication costs as it would be sufficient to merely send the newest w-th measurement instead.

To efficiently handle sliding windows PAO fosters a message suppression strategy to maintain low communication burden, while providing approximate answers of satisfactory quality. In particular, consider a clusterhead which has received $\measuredangle Y_t^{S_u}$ from S_u and assume a parameter ϵ encapsulated in the basestation's inquiry. In the upcoming window, we allow motes to suppress their own messages when $\measuredangle Y_{t_{new}}^{S_u} \in [\measuredangle Y_{t_{previous}}^{S_u} - \epsilon, \measuredangle Y_{t_{previous}}^{S_u} + \epsilon]$, where $\measuredangle Y_{t_{previous}}^{S_u}$ refers to the last value that the mote has transmitted to its clusterhead and $\measuredangle Y_{t_{new}}^{S_u}$ refers to the latest computed (but not necessarily transmitted) $\measuredangle Y_t^{S_u}$ value.

At clusterheads, similarity tests are performed in the same way as before. Nevertheless, the suppression of messages introduces approximate characteristics to PAO. It can easily be observed that for pairs of motes which did not suppress their messages the corresponding test between them will provide exact result. We now outline the cases of accurate similarity estimation despite message suppression:

- For pairs of motes that both suppress their messages clusterheads rely on $\measuredangle Y_{t_{previous}}^{S_u}, \measuredangle Y_{t_{previous}}^{S_v}$ to obtain an answer regarding their similarity. Without loss of generality, we assume that $\measuredangle Y_{t_{previous}}^{S_u} < \measuredangle Y_{t_{previous}}^{S_v}$. When $\measuredangle Y_{t_{previous}}^{S_u} + \epsilon + \measuredangle_{thres} < \measuredangle Y_{t_{previous}}^{S_v} - \epsilon$ the test is always accurate.
- Provided that S_v does not suppress its message while S_u does, clusterheads take into account $\measuredangle Y_{t_{previous}}^{S_u}, \measuredangle Y_{t_{new}}^{S_v}$. Assuming $\measuredangle Y_{t_{previous}}^{S_u} < \measuredangle Y_{t_{new}}^{S_v}$ (the other case is symmetric), a correct answer is ensured when $\measuredangle Y_{t_{previous}}^{S_u} + \epsilon + \measuredangle_{thres} < \measuredangle Y_{t_{new}}^{S_v}$. Otherwise, the result of the test might be either faulty or correct, depending on the actual changes in $\measuredangle Y_{t_{new}}^{S_u}, \measuredangle Y_{t_{new}}^{S_v}$. Obviously, setting $\epsilon = 0$ is equivalent to requiring exact results. Moreover, notice that the above strategy manages to save communication costs irrespectively of the window type. Eventually, we note that ϵ can be dynamically adjusted by motes. Due to space limitations, we omit the corresponding discussion, but we refer interested readers to [4] for further details.

5. FROM PAO TO PAO+

During the intra- and inter-cluster communication phases of our PAO algorithm clusterheads are assigned the load of angle/vector reception as well as the processing burden of similarity test determination. This means that they consume extra power resources during these procedures compared to regular nodes in the clusters. Remaining energy is a primary criterion in any clustering protocol for a mote to be maintained as clusterhead. Thus, the network will need to frequently pause its operation and be led to a reorganization process so as to elect new clusterheads (which also yields an amount of communication cost for sensors). Bearing these, in PAO+ we introduce a hashing mechanism that spreads the intracluster communication and comparison load. Moreover, recall our similarity test $|\measuredangle Y_t^{S_u} - \measuredangle Y_t^{S_v}| \leq \measuredangle_{thres}$. A different reading of the inequality says that sensors with angle differences above $\measuredangle_{thres}$ should not be compared for similarity since we know in hand that the test cannot be successful. Nonetheless, having arrived at a clusterhead comparisons will inevitably be performed even for motes with high angle differences. PAO+ hashing mechanism also manages to prune unnecessary comparisons of motes exhibiting highly dissimilar behavioral patterns.

Load Distribution and Comparison Pruning in PAO+. Apart from electing the clusterhead we choose additional B nodes in the formed cluster as the hashing *Buckets*. To define the hashing procedure we need to clarify: i) the hash key, ii) the hash key space, iii) the hash function application. We proceed by presenting the aforementioned parameters. The hash key is set to be the angle $\measuredangle Y_t^{S_u}$ of mote S_u which means that the hash key space is determined by the fact that $-\frac{\pi}{2} < \measuredangle Y_t < \frac{\pi}{2}$. The previous range is equally distributed between the available buckets such that a bucket B_i holds a range between $[-\frac{\pi}{2} + i \cdot \frac{\pi}{B}, -\frac{\pi}{2} + (i+1)\frac{\pi}{B})$. Next, the hash function that is applied by motes in order to decide the receiver bucket is: $H(\measuredangle Y_t^{S_u}) = \lfloor \frac{\measuredangle Y_t^{S_u}}{\frac{\pi}{B}} + \lceil \frac{B}{2} \rceil \rfloor = B_i$. Thus, in the intracluster processing, instead of letting all regular nodes transmit their data towards the clusterhead we impose that they should be sent to the $\lfloor \frac{\measuredangle Y_t^{S_u}}{\frac{\pi}{B}} + \lceil \frac{B}{2} \rceil \rfloor$-th bucket.

Obviously, the process groups motes with similar trends in buckets while highly dissimilar motes hash in distant buckets in terms of the hash key space assignment. However, at the edges of the buckets similarity may still exist. More precisely, to guarantee that a node can be witnessed by any similar within the cluster, $\measuredangle Y_t^{S_u}$ needs to be sent to the bucket nodes that cover the range $\lfloor \frac{\max\{\measuredangle Y_t^{S_u} - \measuredangle_{thres}, 0\}}{\frac{\pi}{B}} + \lceil \frac{B}{2} \rceil \rfloor$ to $\lfloor \frac{\min\{\measuredangle Y_t^{S_u} - \measuredangle_{thres}, \pi\}}{\frac{\pi}{B}} + \lceil \frac{B}{2} \rceil \rfloor$. Utilizing more buckets reduces the range of each, but results in more $\measuredangle Y_t^{S_u}$s being transmitted to multiple buckets. PAO+ selects the value B (whenever at least B nodes exist in the cluster) by setting $B < \frac{\pi}{\measuredangle_{thres}}$ which limits the number of bucket nodes to which motes transmit their data to the range $\lfloor \frac{\max\{\measuredangle Y_t^{S_u} - \measuredangle_{thres}, 0\}}{\frac{\pi}{B}} + \lceil \frac{B}{2} \rceil \rfloor$ to $\lfloor \frac{\measuredangle Y_t^{S_u}}{\frac{\pi}{B}} + \lceil \frac{B}{2} \rceil \rfloor$. The latter range is guaranteed to contain at most 2 buckets.

In order to make sure that the similarity test is not performed more than once we impose the following rules: (a) For $\measuredangle Y_t^{S_u}$s

mapping to the same bucket, the similarity test between them is performed only in that bucket node; and (b) For $\measuredangle Y_t^{S_u}$s mapping to different bucket nodes, their similarity test is performed only in the bucket node with the lowest B_i.

Each bucket reports the set of outliers that it detected, along with their support, to the clusterhead. Any mote reported by at least one, but not all buckets to which it was transmitted, is guaranteed not to be an outlier, as it has reached $minSup$ at some bucket. Even when a mote is reported by all the buckets it was hashed, the support that its measurements have gained is distributed between buckets needing to be summed up. Hence, the received support is added, and only those $\measuredangle Y_t^{S_u}$s that did not receive the required overall support, from the buckets they were hashed to, are considered outliers.

We note that the whole process does not change the organization of the data as presented in Section 4, as it simply introduces an extra fragmentation step on the data. Eventually, sensor nodes that do not manage to pass the linearity test are assigned to a separate bucket to be compared with each other, omitting the hashing mechanism.

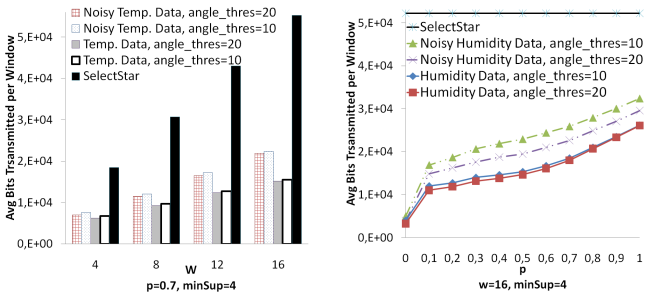
Load Balance in PAO+. The hashing mechanism we discussed distributes the processing and communication load during the intracluster phase of our algorithm. However, it does not guarantee that load apportionment will be equal between buckets. A naive way to confront occasions of high unbalanced load is to let buckets locally redetermine the hash key space for themselves and simply route any hashed information outside the new range to other left/right neighboring buckets. However, in this case our primary concern regarding bandwidth preservation is violated which at last deteriorates the power consumption for those buckets.

To balance the load amongst buckets and simultaneously achieve an efficient way to do so, PAO+ takes into consideration the monitored trends' distribution. More precisely, PAO+'s load balancing mechanism acts after the initial hash key space assignment and involves the construction of simple equi-width histograms. As buckets receive data from other motes in the cluster they maintain frequency counts of $\measuredangle Y_t^{S_u}$s. Subsequently, each bucket communicates to its clusterhead the estimated frequency counts along with the width parameter used in their construction. Every clusterhead is aware of the current hash key space assignment since it took part in the previous partitioning and can easily reconstruct the histograms.

Finally, based on these compact representation of the monitored patterns distribution, a new key space allocation is determined and broadcasted to all nodes in the cluster. The whole procedure can be periodically repeated i.e every a number of $w \cdot e$ time intervals to allow adaptations to changing data distributions.

6. EVALUATION RESULTS

Experimental Setup. In order to perform a comprehensive study of our algorithms varying different parameters, we developed a custom simulator. We randomly located sensors in a rectangular area and set the packet size to 16 bytes. We tested our PAO and PAO+ algorithms using a real world data set termed Intel Lab Data. The data set consists of temperature and humidity measurements collected by 48 motes for a period of 633 and 487 epochs, respectively, in the Intel Research, Berkeley lab [5]. To test our methods in harsh conditions, apart from using the original data sets, we produced extra versions (termed as "Noisy" in our experiments) where we increased the complexity of the measurements by specifying for each mote a 6% probability that it will fail dirty at some epoch. Failures were simulated using a known deficiency [5] of the MICA2 temperature sensor according to which a mote that fails-dirty increases its samples until it reaches a maximum value. We set that increment to 1 degree per epoch with a maximum value of 100 degrees. To prevent the measurements from lying on a straight line, we also impose a noise up to 15% at the values of a node that fails dirty. Additionally, each node with probability 0.4% at each epoch obtains



(a) Avg Bits Transmitted vs w (b) Avg Bits Transmitted vs p

Figure 3: Avg Bits Transmitted per Window varying w and p

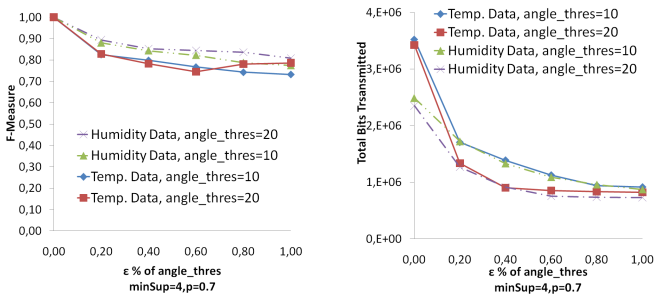
a random, spurious reading between 0 and 100 degrees. We organized our network in four clusters and utilized a $minSup$ value of 4 (i.e 1/3 of the total motes in a cluster). Eventually, in each experiment we used $\measuredangle_{thres}$ of 10 and 20 degrees which account for rigid and more relaxed cases of outlier definitions.

Bandwidth Consumption in PAO. We first present a set of experiments regarding the regular operation of our framework using disjoint time windows. We compare the bandwidth consumption of PAO against a centralized method termed "SelectStar" that collects all data in the query source and performs the outlier detection process there, instead of using PAOs in-network paradigm.

Figure 3(a) shows the reduction in bandwidth consumption provided by PAO for the temperature data, when varying the window size w, for a $p = 0.7$ threshold. PAO manages to reduce the average amount of transmitted data per window up to a factor of 1/3.8 for the original and 1/2.6 for the noisy data compared to the SelectStar approach. Additionally, Figure 3(b) depicts the average communication preservation depending on p's strictness for the humidity data using $w = 16$. Please note that setting the window size to the maximum of the previously (Fig. 3(a)) cited windows constitutes a worst case scenario for PAO, as the larger the window the fewer the motes that manage to adapt to the linear model. We can observe that the gains in the average amount of transmitted bits range between 1/1.65 and 1/15 for $p = 1$ and $p = 0$, respectively (SelectStar is the straight line at the very top of the figure). Notice that setting $p = 1$ for the noisy data version results in the transmission of all Vec_{S_u}s since no mote satisfies that threshold in the examined data sets. As a result, the aforementioned lower bound of 1/1.65 expresses the worst case gains solely provided by PAOs in-network outlier detection approach.

Sliding the Window. We now investigate the characteristics attributed to PAO when operating over sliding windows, thus approximately pinpointing outlying values and reducing the communication burden by suppressing messages as described in Section 4. Figures 4(a), 4(b) present the accuracy of our framework and the amount of communicated bits for different ϵ values expressed as a percentage of the specified $\measuredangle_{thres}$. We compute the accuracy of PAO using the $F{-}measure = 2/(1/Precision + 1/Recall)$ metric where precision specifies the percentage of reported outliers that are true outliers, while recall specifies the percentage of outliers that are reported by our framework. Notably, PAO exhibits high accuracy with F-measure values $\geq 80\%$ in most of the cases while managing to reduce the total amount of communicated data to $1/3.6$ on average compared to the mere transmission of the latest value in the window (for $\epsilon = 0$).

Eventually, Figure 5 shows the corrected answers that would be obtained upon utilizing PAO on par with a simple aggregate (max) query. The parameters used during the outlier detection are included in the figure. In each epoch, we initially computed the maximum humidity reported by the network using the original data sets (AGGREGATION). Then we posed the same aggregate query and



(a) Accuracy varying ϵ (b) Total Bits Transmitted vs ϵ

Figure 4: Approximate Processing over Sliding Window, w=16

let it be executed after the outlier detection and removal performed by PAO using $\epsilon = 5$ degrees. We can observe that PAO manages to prevent the distortion on the final results caused by the outliers for the vast majority of the epochs.

PAO+ Leverages. To validate the ability of the hashing mechanism introduced by PAO+ to distribute the intracluster load, as well as to prune unnecessary comparisons, in Table 1 we present the effect of bucket node introduction utilizing disjoint time windows of $w = 16$ size. We used different cluster sizes between 24 and 48 motes while varying the number of buckets from 1 to 4. Moreover, the notation "+1" used in the number of buckets expresses the utility of an additional bucket for motes that do not succeed in the linearity test. The table provides average results per window. In particular, we include the average number of comparisons (Cmps) that take place in a window along with the average number of motes that sent their data to 2 buckets (Multihashes). Furthermore, we present the average hashes received by a bucket (Hashes Per Bucket). It can easily be deduced that increasing the number of buckets dramatically reduces the number of performed comparisons which validates the usefulness of PAO+ in this particular aspect. On the other hand, the number of multihashes and the number of hashes per bucket regard a transmission cost mainly charged to cluster's regular motes and the load distribution between buckets, correspondingly. The adoption of more buckets, causes an increase in multihashes and a simultaneous decrease in the number of hashes per bucket. This is interpreted as a shift in the energy consumption from clusterhead and bucket nodes to regular cluster motes caused by the increment of bucket nodes' number. Achieving appropriate balance aids in keeping intracluster, uniform energy consumption, which subsequently leads to infrequent network reorganization.

7. CONCLUSIONS

In this paper we presented PAO, an outlier detection framework that manages to perform over multiple window types and allows users to choose between exact or approximate operation. We also devised PAO+'s mechanisms that manage to prune unnecessary comparisons and balance the intracluster load during the outlier detection process. Our experimental evaluation using real world datasets validated that our framework can pinpoint outlier readings ensuring significantly decreased amount of communicated information. It also showed the ability of approximate PAO to provide results of high quality with further reduced bandwidth consumption.

8. REFERENCES

[1] P. Andreou, D. Zeinalipour-Yazti, A. Pamboris, P. K. Chrysanthis, and G. Samaras. Optimized Query Routing Trees for Wireless Sensor Networks. *Information Systems*, to appear, 2010.

[2] S. D. Bay and M. Schwabacher. Mining Distance-based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule. In *KDD*, 2003.

[3] J. Chen, S. Kher, and A. Somani. Distributed Fault Detection of Wireless Sensor Networks. In *DIWANS*, 2006.

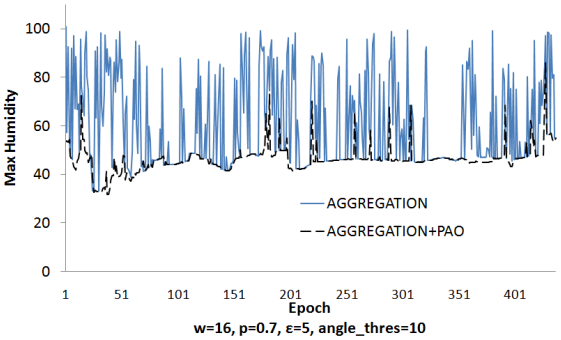


Figure 5: Max Humidity Values after Outlier Removal

		Δ_{thres}					
		10			20		
Cluster Size	Buckets	Cmps	Multi-hashes	Hashes Per Bucket	Cmps	Multi-hashes	Hashes Per Bucket
24	1+1	70.45	0	12	70.46	0	12
	2+1	33.72	0.78	8.26	35.88	1.63	8.54
	4+1	16.58	2.21	5.24	18.39	4.65	5.73
36	1+1	160.27	0	18	160.41	0	18
	2+1	77.61	1.22	12.41	81.63	2.12	12.71
	4+1	37.38	3.39	7.88	42.26	7.05	8.61
48	1+1	286.38	0	24	286.80	0	24
	2+1	137.39	1.63	16.54	145.94	3.04	17.01
	4+1	67.39	4.53	10.51	75.33	9.12	11.42

Table 1: Bucket Introduction in PAO+ (w=16, p=0.75)

[4] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical In-Network Data Aggregation with Quality Guarantees. In *EDBT*, 2004.

[5] A. Deligiannakis, Y. Kotidis, V. Vassalos, V. Stoumpos, and A. Delis. Another Outlier Bites the Dust: Computing Meaningful Aggregates in Sensor Networks. In *ICDE*, 2009.

[6] E. Elnahrawy and B.Nath. Cleaning and querying noisy sensors. In *WSNA*, 2003.

[7] K. Georgoulas and Y. Kotidis. Random Hyperplane Projection using Derived Dimensions. In *MobiDE*, 2010.

[8] N. Giatrakos, Y. Kotidis, A. Deligiannakis, V. Vassalos, and Y. Theodoridis. TACO: Tunable Approximate Computation of Outliers in wireless sensor networks. In *SIGMOD*, 2010.

[9] Y. j. Wen, A. M. Agogino, and K.Goebel. Fuzzy Validation and Fusion for Wireless Sensor Networks. In *ASME*, 2004.

[10] S. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Pervasive*, 2006.

[11] B. Karp and H. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *MOBICOM*, 2000.

[12] Y. Kotidis, A. Deligiannakis, V. Stoumpos, V. Vassalos, and A. Delis. Robust Management of Outliers in Sensor Network Aggregate Queries. In *MobiDE*, 2007.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.

[14] M. Qin and R. Zimmermann. VCA: An Energy-Efficient Voting-Based Clustering Algorithm for Sensor Networks. *J.UCS*, 13(1), 2007.

[15] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation. In *MobiDE*, 2003.

[16] B. Sheng, Q. Li, W. Mao, and W. Jin. Outlier detection in sensor networks. In *MobiHoc*, 2007.

[17] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online Outlier Detection in Sensor Data Using Non-Parametric Models. In *VLDB*, 2006.

[18] X. Xiao, W. Peng, C. Hung, and W. Lee. Using SensorRanks for In-Network Detection of Faulty Readings in Wireless Sensor Networks. In *MobiDE*, 2007.

[19] O. Younis and S. Fahmy. Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach. In *INFOCOM*, 2004.

[20] D. Zeinalipour, P. Andreou, P. Chrysanthis, G. Samaras, and A. Pitsillides. The Micropulse Framework for Adaptive Waking Windows in Sensor Networks. In *MDM*, 2007.

[21] Y. Zhang, N. Meratnia, and P. Havinga. Outlier detection techniques for wireless sensor networks: A survey. *International Journal of IEEE Communications Surveys and Tutorials*, 12(2), 2010.

[22] Y. Zhuang, L. Chen, S. Wang, and J. Lian. A Weighted Moving Average-based Approach for Cleaning Sensor Data. In *ICDCS*, 2007.

Future Directions in Sensor Data Management: A Panel Discussion

Demetris Zeinalipour
University of Cyprus, Cyprus
dzeina@cs.ucy.ac.cy

ABSTRACT

We will soon celebrate 10 years of research and development in the area of sensor networks. During this decade, we have witnessed the emergence of specialized embedded systems, operating systems, data-oriented management systems as well as programming languages for ad-hoc monitoring of the environment at a high fidelity. All the advances have brought us one step closer to the initial Smartdust vision. The first signs of data management approaches to cope with the inherent complexities of sensor networks arose in 2003, with the release of prototype database systems and the spin off of specialized research conferences (i.e., IPSN in 2003) and workshops (i.e., DMSN in 2004).

In the recent years, we have been witnessing a paradigm shift from the initial target of sensor networks, which focused on low-power embedded sensing devices utilized for environmental and habitant monitoring, to new domains involving more powerful devices (such as smartphone devices) and applications (such as people-oriented social networking applications). We have also been witnessing the emergence of complementary technologies such as stream processors, cloud data analytic frameworks, semantic web technologies and others. Although many of these frameworks have similar assumptions and goals, it is not clear how these can drive or be driven in the future by sensor data management research.

The aim of this panel is to discuss: (1) to what extend the vision of applying data management techniques to sensor network research has been successful over the years (e.g., adoption of ideas proposed by the community); ii) to examine the significance of recent advances and to identify new directions that can foster research in sensor data management.

DMSN '10, September 13, 2010, Singapore

List of Panelists :

- Yanlei Diao (University of Massachusetts Amherst, USA)
- Le Gruenwald (National Science Foundation, USA)
- Christian S. Jensen (Aarhus University, Denmark)
- Kian-Lee Tan (National University of Singapore, Singapore)