Computing and Informatics, Vol. 30, 2011, 1181–1199

A TREE-BASED HIERARCHY DATA STORAGE FRAMEWORK IN A PERVASIVE SPACE

Zhuzhong Qian

State Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University
210093 Nanjing, P. R. China
e-mail: qzz@nju.edu.cn

Ilsun You*

School of Information Science
Korean Bible University
South Korea
e-mail: isyou@bible.ac.kr

Youyou Lu, Sanglu Lu

State Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University
210093 Nanjing, P. R. China
e-mail: Luyouyou87@gmail.com, sanglu@nju.edu.cn

Abstract. Context data is important information for catching the behaviors of applications in a pervasive space. To effectively store huge amount of data, tree-like layered storage architecture is proposed, where the leaf nodes collect data from sensing devices. In order to integrate data from mobile devices, the related leaf nodes that get data from the same device should upload and store the data to

* corresponding author

the *host node*. This paper presents a deep study of the data storage problem and proposes a global algorithm GHS and an online algorithm DHS to dynamically select the host node, which reduces the communication cost significantly. This paper also gives the theoretical and experimental analysis of these algorithms, which shows both GHS and DHS are correct and effective.

Keywords: Host selection, distributed storage, pervasive space

1 INTRODUCTION

With the development of mobile devices and wireless communication, pervasive computing becomes a hot spot in both academic and industry areas. In a pervasive computing environment, sensing and monitoring equipments are deployed to collect context data, which is used to recognize user activities, collect resource status, etc. These devices (e.g. body sensors, RFID tags, wireless cameras) are small, cheap and widely deployed. Based on them, applications can track or predicate the locations and status of objects. For example, any object in the physical world can be marked by an RFID tag with a global unique ID and they can be connected with each other to be a network called "Internet of Things" [2]. SpotON [1] utilizes the RFID technology to locate an object by computing the strength of signals from several base stations. Body Sensor Network (BSN) is discussed in [2] to implement smart homes. Furthermore, some context-aware system can dynamically change their behaviors to adapt the changing of the environment by sensing the context, and related technology is proposed to achieve it effectively. [3]

Currently, the memory size of sensor devices is becoming larger and the storage-centric sensor network also has been widely discussed [4, 5, 6, 7], which argues that some of the real time data should be stored locally in the sensing devices. However, even in such a network, the sensor node cannot restore all the context data, they have to push the historical data to the servers from time to time. In the meantime, in order to synthetically analyze the system status, we need to integrate the context from different servers. In order to effectively manage the huge amount of data, the layered data storage approach is proposed and widely discussed, where several data servers are organized in a tree-like architecture and each leaf node collects context data from a number of sensing devices within its communication domain.

In such an architecture, only the leaf nodes communicate directly with the sensing devices and one sensing device sends context data to one leaf node at a certain time point. Then, the context data will be uploaded to the up-layer nodes based on some policies [7]. However, several types of sensing devices such as body sensors and RFID tags are moving with the attached objects. Because of the communication areas of these devices are limited, the context data generated by them would be sent to the leaf nodes that are closed to the devices, i.e., one mobile device may send data to different nodes as it moves on. In a simple layered architecture, to integrate the context data generated from the same device, we have to upload the data to the data server that can cover all the leaf nodes which have communicated with the device. Figure 1 is an example of the storage system. As the device moves, the nodes 11, 12 and 5 may maintain part of context data from this sensing device. Because only node 0 could cover all these three nodes, the data integration has to be implemented on the root (node 0).

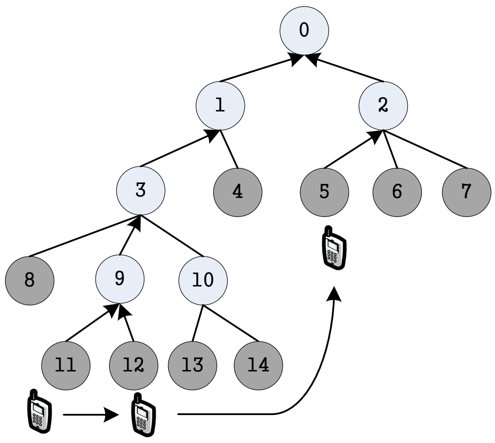


Fig. 1. The layered data storage architecture

Generally, a mobile device usually moves among a certain domain and communicates with the corresponding leaf nodes in this domain. For example, RFID tag attached with a student usually communicates with the readers in classrooms, dormitories, and cafeterias. If we choose a host node that is close to these leaf nodes to integrate the context data, it may reduce communication cost. Furthermore, in the traditional simple layered architecture, as the mobile device moves to a new domain by accident, the integrate host node may be changed.

In this paper, we present a deep study of the distributed context data storage for mobile sensing devices and propose host selection algorithm to evaluate and select a host that would reduce the communication cost, which is an extension of the paper [8]. The main contributions are:

1. we present a deep study of the layered data storage architecture and give the data storage and query policies;
2. we present a centralized algorithm GHS to choose the host;
3. an on-line dynamic host selection algorithm DHS is proposed to adapt for a more flexible situation;
4. we verify the performance of the host selection algorithms in both theoretical and experimental analysis.

The rest of this paper is organized as follows. Section 2 presents some related work. Section 3 gives a brief introduction of the system including the context data storage and query. Section 4 is the detailed description of the global host selection algorithm and proves its correctness. Section 5 gives the dynamic host selection algorithm DHS. Section 6 shows the simulation results and discusses the related issues. Section 7 concludes the paper.

2 RELATED WORK

In a pervasive environment, sensing devices such as RFID tags, sensors are widely deployed. How to effectively store and query context data is a challenge for huge amount of context data management.

Gonzalez proposes a new warehousing model that preserves object transitions while providing significant compression and path-dependent aggregates [9]. In [10], an EPC encoding scheme using bitmap data type is presented to compress the storage of EPC tag collections.

A good data storage framework is a good method to improve the performance. Paper [11] divides framework into four categories: unstructured P2P, structured P2P, metadata integration centralized and data integration centralized architecture. In supply chain management, EPC global [12], an industry driven standard group raises three layers of data exchange standards: Physical Object Exchange Standards, Infrastructure Standards for Data Capture and Data Exchange Standard. Intra-enterprise data exchange follows the infrastructure standards to define the interaction interfaces for data access of application. Following the data exchange standard, inter-enterprise data exchange relies on EPC Discovery Services, ONS (Object Naming Service) and other core services. Paper [13] also proposes a query processing technique with combination of P2P to improve data sharing performance in tracking and tracing. Moreover, [14, 15] present data exchange by web services. In vehicle tracking application, [16] presents HERO (Hierarchy Exponential Region Organization), in which hierarchy structure with restrict location updating policy is built up, to ensure real time and network traffic needs.

Most works are designed for specific applications. However, data repository framework for one application does not fit for the others due to specific patterns in specific applications. For example, the transport of goods in supply chain is sequentially delivered from manufacturers, distributors and retailers, while vehicle runs randomly in the whole city. Web services for data exchange used in supply chain do not work well for vehicle tracking, and vice versa.

Currently, the storage centric sensor network is proposed to store part of the data in the local node and upload the rest of the data to the upper node. The upper layered node also restores part of the data in its local area and uploads the rest to the upper node till the root node. This kind of storage strategy is just a tree-like data storage architecture. However, these works focus on the data extract policy and integrate all the context data in the root which makes the root be the bottleneck of the system.

This paper presents a deep study of layered data storage problem and investigates the host node selection problem to effectively restore the context data.

3 SYSTEM OVERVIEW

3.1 Layered Data Storage Architecture with Dynamic Host

In order to monitor the status of the pervasive computing environment, the sensing devices are widely deployed to collect the context data and send the context data to the server which is the leaf node in the layered context data storage system. Then, the data will be uploaded to the up-layer servers with a certain policy which is related to different applications. It is worth mentioning that the tree structure is an overlay storage framework, where each node could be matched to one or more real data servers. Some mobile devices which have more power and memory also could be a server node.

In this paper, we propose a dynamic host selection strategy that selects the "closest" host for the device, instead of the node covering all the related leafs. Here, the "closest" means the host is close to the most hot spots of the sensing device. In Figure 1, we may select node 9 as the host if the device always sends data to node 11 and 12 while only few data is sent to node 5.

3.2 Data Storage and Query Based on Dynamic Host

Once the host for one device is selected, it is necessary to inform the ancestor until the root. All the data from this device should be transformed to its host node.

1) **Context Data Collection.** In the layered storage system, all the sensing devices only contact with the leaf nodes and send context data to them. Because both sensing device and leaf node have certain communication areas, so each device may communicate with few certain leaf nodes at a time. In this paper, we assume that the communication area of all the leaf nodes can cover the whole system, so that all sensing devices can communicate with at least one leaf node at a time. If the sensing device can communicate with more than one leaf nodes at a time, it will choose the best one based on the communication quality. Once the communication between a pair of devices and leaf node is set up, the device will periodically send context data to the leaf node. Each leaf node maintains a *data frequency list* (DFL), which records the number of data received from different devices.

2) **Host Selection.** In this storage system, each storage node in the tree has entire host index information of its descents. The format of the index is (Device_ID, Node_ID, next_hop). That is, if one of its descent node (say node 4) is the host

of device A, it has the index information (A, 4, (*its parent node*)) The root of the whole storage system maintains an index of the complete context data storage information. When the host of a sensing device is selected, the host node should send an index message to its parent node which will continue to send this message to the upper layer node until the root. When the device moves to a new domain, the host may shift to a new host. In this case, the new host will send a new index message to its parent and this message will be forwarded until the root. And the old host will send an update message to its up-layered nodes to delete the old index. Then, the data stored in the old host will be transferred to the new host.

3) **Data Storage.** Based on the above description, all the nodes may have indexes of all the host information of their descents. Thus, the root has all the host information of the whole system. Generally, the node which obtains the context data is the descent node of the host of the device, and it maintains the host information. Accordingly, when the context data is sent to these nodes, the data will be uploaded to their upper-layered nodes layer by layer until the host node. However, if the device moves to a new domain, things may be more complicated. The leaf node that obtains the data may not have the host information of this device. In this case, this node will send the data to its parent which will forward the data to its up-layered node until to the node that has the host information of the device. Then, this node will transform the data to the node according to the next_hop, i.e. one of its children nodes.

4) **Data Query.** Context data query request could be from any node. For any query request, the system analyzes the device ID of the request data and finds out the host of the related device to get the context data. The query procedure is similar with the storage procedure. That is, if the node has the index of the host information, it forwards the request to the next_hop, otherwise it will forward the request to its parent node.

3.3 Host Selection

The goal of host selection is to reduce the communication cost. This strategy is based on the fact that a certain mobile device sends data to the different leaf nodes with different frequencies. For example, the trace of a body sensor is exactly the owner's trace, and as we know, people mainly appear in the working place and home. Thus, choosing a node that is close to the leaf nodes in these two domains may save a lot of communication resources.

For any mobile device, all leaf nodes may have the chance to communicate with the device and get data. When the host is selected, the storage can be converted into a new overlay tree with the host node as the root, which is called context storage tree (CST). This context storage tree is a logic storage structure, it only shows how the context data of a certain device is delivered from the leaf nodes to the host. Each device has its own storage tree, although they have the same physical storage architecture. Based on the physical storage architecture in Figure 1, its CST can be described as Figure 2, if the node 10 is selected as the host.

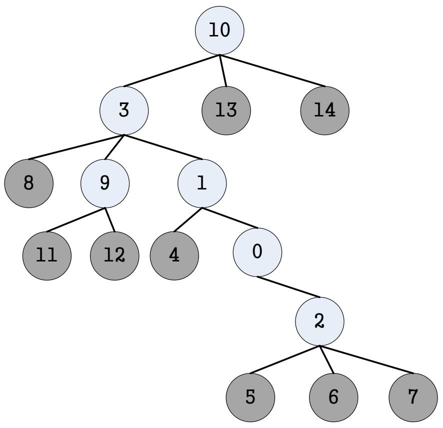


Fig. 2. Context storage tree

For a mobile device, suppose we know the probabilities that it sends data to the different leaf nodes. The probabilities are denoted as the weight (w) of the leaf nodes. Concretely, we get the weight from the data frequency list, i.e. we use the number of communication between the device and the node to represent the frequency. Thus, the weight is monotonic value increase as the time goes on. Then, the storage cost can be represented as the weighted path length (WPL) of the tree. Thus, the host selection problem can be represented as constructing the context storage tree with the minimum WPL.

Definition 1 (Host selection)**.** Given a tree-like context data storage system, let WPL(i) represent the weighed path length of the context storage tree with the node i as root. The host selection problem is to find a node j, satisfy WPL(j) = min(WPL(i), $i \in V$), where V is the set of all the nodes in the system. The storage tree with the root V_j is called optimized storage tree.

4 GLOBAL HOST SELECTION

4.1 The Algorithm

In this algorithm, we introduce two variants: WPL and leafWeight. WPL represents the weighted path length while leafWeight is the sum of the probability that all the descendant leaf nodes communicate with the device.

Initially, the leafWeight of each leaf node is got from historical data, and all the WPL and leafWeight of the non-leaf node are set as 0. Then, the nodes with least leafWeight are selected from current leaf node set, and update WPL and leafWeight of its neighbor with the following equation:

$$\text{WPL} \mathrel{+}= \text{WPL\_delete} + \text{leafWeight\_delete} \tag{1}$$

$$\text{leafWeight} \mathrel{+}= \text{leafWeight\_delete} \tag{2}$$

This process continues until one node is left, and this node is the root of the storage tree. The pseudocode of GHS is given in Algorithm 1.

Algorithm 1 Global Host Selection Algorithm

```
1: for each leaf node, WPL[$i$] = 0, LeafWeight[$i$] = $p[i]$
2: for each non-lead node, WPL[$i$] = 0, LeafWeight[$i$] = 0
3: LeafSet = $B$
4: while LeafSet has more than one node do
5:     Find leaf node $l[j]$ from LeafSet with least leafWeight[$j$]
6:     $l[i]$ is neighbor (parent or child) of $l[j]$
7:     WPL[$i$] = WPL[$i$] + WPL[$j$] + leafWeight[$j$]
8:     leafWeight[$i$] + = leafWeight[$j$]
9:     Remove $l[j]$ from LeafSet
10: end while
11: Host node is the left node of LeafSet
```

4.2 The Proof of Correctness

In context storage tree (CST), w_n is the weight of node v_n. leafWeight is the communication probability for leaf node, and is the probability sum of all descendant leaf nodes for non-leaf nodes, which is initialized as 0 and updated by the leaf node deletion.

Let WPL_n the weighted path length of subtree with v_n as the root, and WPL'_n the weighted path length of reconstructed CST with v_n as the root. WPL_n is computed with $\text{WPL}_n = \sum_{v_i \in \text{desendant}(v_n)} (w_i * l_i)$.

Theorem 4.1. In a tree,for non-leaf node v_m,

$$\text{WPL}_m = \sum_{v_i \in \text{children}(v_m)} \left(\text{WPL}_i + \sum_{v_j \in \text{desendant}(v_i)} w_j \right). \tag{3}$$

Proof. Let PL_{ij} be the length of the path from node v_j to node v_i. For any child node v_n of non-leaf node v_m, the subtree with root v_n is denoted as sub(v_n).

$$\sum_{v_j \in \text{sub}(v_n)} (w_j * PL_{mj}) = \sum_{v_j \in \text{sub}(v_n)} (w_j * (PL_{nj} + PL_{mn})) = \text{WPL}_n + \sum_{v_j \in \text{sub}(v_n)} w_j \tag{4}$$

Therefore, the weighted path length of subtree with node v_m as root is

$$\mathrm{WPL}_m = \sum_{v_i \in \mathrm{children}(v_m)} \sum_{v_j \in \mathrm{sub}(v_i)} (w_j * PL_{mj}) = \sum_{v_i \in \mathrm{children}(v_m)} \left(\mathrm{WPL}_i + \sum_{v_j \in \mathrm{sub}(v_i)} w_j \right) \tag{5}$$

□

From Theorem 4.1, we can get the relation of one node's WPL with his children's WPLs, which is used to compute the WPL of any subtree in the CST recursively.

Theorem 4.2. Let L denote the leaf set. If $w_m = \min_{v_i \in L} w_i$, the CST with root v_m has $\mathrm{WPL}'_m \geq \min_{v_i \in V} \mathrm{WPL}'_i$.

Proof. $DL(v_i) = \mathrm{desendant}(v_i)L$. The CST with root v_m has

$$\mathrm{WPL}'_m = \mathrm{WPL}_m + \mathrm{WPL}_n + \sum_{v_i \in \mathrm{sub}(v_n)} w_i. \tag{6}$$

The CST with root v_n has

$$\mathrm{WPL}'_n = \mathrm{WPL}_n + \mathrm{WPL}_m + w_m. \tag{7}$$

For $v_q \in \mathrm{DL}(v_n)$, $\sum_{v_i \in \mathrm{sub}(v_n)} w_i \geq w_q$, as $w_m = \min_{v_i \in L} w_i$, $w_q \geq w_m$, comparing the above two inequalities, we get

$$\sum_{v_i \in \mathrm{sub}(v_n)} w_i \geq w_m \tag{8}$$

and,

$$\mathrm{WPL}'_m - \mathrm{WPL}'_n = \sum_{v_i \in \mathrm{sub}(v_n)} w_i - w_m \geq 0. \tag{9}$$

Therefore, $\mathrm{WPL}'_m \geq \min_{v_i \in V} \mathrm{WPL}'_i$. □

Theorem 4.2 shows that there is another node other than the deleted leaf node having a less or equal WPL. That is, the WPL of the CST with deleted leaf node as its root is not less than the minimum WPL. After the iterative deletion, the node which CST has a minimum WPL remains, and this node is the best candidate for the host node.

Theorem 4.3. The deletion of leaf node does not change the WPL of CST. Let the weighted path length of CST before and after deletion operation be WPL and WPL$'$ respectively; we have $\mathrm{WPL}' = \mathrm{WPL}$.

Proof. In a deletion operation, v_m is the node to be deleted and v_n is its neighbor. The weighted path length and weight of subtree with root v_i before deletion operation are WPL_i, w_i and WPL'_i, w'_i respectively.

For $\forall v_i \in V, v_i \neq v_m$, if v_i is the root of Data Repository Spanning Tree, denoted as v_0. There is a path from v_0 to v_m, denoted as $(v_0, v_1, v_2, \ldots, v_k, v_n, v_m)$.

$$\begin{aligned}
\mathrm{WPL}' - \mathrm{WPL} &= \mathrm{WPL}'_0 - \mathrm{WPL}_0 \\
&= \sum_{v_i \in \mathrm{children}(v_0)} \left(\mathrm{WPL}'_i + \sum_{v_j \in \mathrm{sub}(v_i)} w'_j \right) \\
&\quad - \sum_{v_i \in \mathrm{children}(v_0)} \left(\mathrm{WPL}_i + \sum_{v_j \in \mathrm{sub}(v_0)} w_j \right) \\
&= \mathrm{WPL}'_1 - \mathrm{WPL}_1 + (w'_n - w_n - w_m) \\
&= \mathrm{WPL}'_1 - \mathrm{WPL}_1
\end{aligned} \tag{10}$$

Similarly, $\mathrm{WPL}'_1 - \mathrm{WPL}_1 = \ldots = \mathrm{WPL}'_k - \mathrm{WPL}_k = \ldots = \mathrm{WPL}'_n - \mathrm{WPL}_n$.

After deletion, node v_n has an equivalent weight path length, $E(\mathrm{WPL}'_n)$. If the equivalent weight path length before deletion is denoted as $E_0(\mathrm{WPL}_n)$, the incremental of equivalent weight path length through update operation is

$$E(\mathrm{WPL}'_n) = \mathrm{WPL}_m + w_m \tag{11}$$

Then,

$$\mathrm{WPL}'_n = E_0(\mathrm{WPL}_n) + \nabla E(\mathrm{WPL}'_n) + \sum_{v_i \in (\mathrm{children}(v_0) - v_m)} (\mathrm{WPL}_i + \sum_{v_j \in \mathrm{sub}(v_i)} w_j). \tag{12}$$

According to Theorem 4.1,

$$\mathrm{WPL}_n = E_0(\mathrm{WPL}_n) + \sum_{v_i \in (\mathrm{children}(v_n) - v_m)} (\mathrm{WPL}_i + \sum_{v_j \in \mathrm{sub}(v_i)} w_j) + (\mathrm{WPL}_m + w_m). \tag{13}$$

From the above two equations, we have $\mathrm{WPL}' = \mathrm{WPL}$. □

Theorem 4.3 proves that no matter which node is selected as the host node, the leaf deletion operation does not change the WPL of the context storage tree, though it changes the structure of the Data Repository Spanning Tree. This guarantees that the cost of the deleted leaf node is equally converted and added to the cost of its neighbor.

In GHS algorithm, each step deletes the minimum weighted leaf node, except when there is another node whose CST has less or equal WPL according to Theorem 4.2. During the deletion operation, the weight of the deleted leaf node is equally converted and added to the weight of its neighbors, and thus it does not change the WPL of the context storage tree according to Theorem 3. The neighbor of the deleted leaf node may become a new leaf node if it has only one neighbor after the deletion operation. This guarantees that GHS algorithm can choose and delete one leaf node each time. After finite steps, the algorithm terminates and only one node is left, for that the number of nodes is finite. The last node is the expected host.

5 DYNAMIC HOST SELECTION

In some pervasive systems, the probabilities of the device sending data to a leaf node are changing from time to time. We cannot determine the weight of the leaf nodes and the host also should be changed as the device moves. In this case, we propose an on-line host selection algorithm to dynamically select the host and optimize the host as the device moves on.

The DHS algorithm is composed of two parts:

1. leaf node weight update and
2. host selection.

If the weight changes, the leaf node sends the increased number of communication $\bigtriangledown w$ to its parent node, the parent node updates the weighted path length as WPL = WPL + $\bigtriangledown w$, and the weight is updated as $w = w + \bigtriangledown w$. After that, the parent node will send the update information to its upper layered node until to the host node.

Algorithm 2 WeightUpdateProcedure

```
1: Recieve $\langle \text{deltaWeight}, \text{deltaWPL} \rangle$ from child node;
2: $w+=\text{deltaWeight}$;
3: $\text{WPL}+=\text{deltaWPL}+\text{deltaWeight}$;
4: Send $\langle \text{deltaWeight}, \text{deltaWPL+deltaWeight} \rangle$ to parent node in current spanning
   tree
```

The host selection will be triggered after the weight is updated.

Algorithm 3 HostReselectionProcedure

```
1: Select node $s$ from children set of $r$ with maximal weight, current node $r$ is root
   of current spanning tree;
2: **if** $(2 * w_s - w_r > 0)$ **then**
3:    $\text{WPL}_r = \text{WPL}_r - \text{WPL}_s - w_s$;
4:    $w_r = w_r - w_s$;
5:    $\text{WPL}_s = \text{WPL}_s + \text{WPL}_r + w_r$;
6:    $w_s = w_s + w_r$;
7:    Set node $s$ as the new host of tree;
8:    Send ReselectHost to node $s$ for reselect;
9: **else**
10:   host $:= r$, that is current node $r$ is host;
11: **end if**
```

The host V_r selects one of its children nodes with the greatest weight and calculates $\bigtriangledown = w_r - 2w_s$. If delta is less than 0, then V_s is selected as the new host. Then, the new host will do the same procedure as the original host until the delta is larger than 0 or to the leaf node. Finally, the host node would be the new root of the storage tree.

5.1 The Proof of Correctness

In the current storage tree with root V_m, the weighted path length is denoted as WPL_m. For the restructured storage tree with root V_n, the weighted path length is denoted as WPL'_n.

Theorem 5.1. In a storage tree with root V_m, V_n is one of the children nodes; if $\nabla_m = (2 * w_n - w_m) > 0$, then $\mathrm{WPL}'_n < \mathrm{WPL}_m$.

Proof. If V_n is the root of the restructured storage tree, then the weighted path length is, $\mathrm{WPL}'_n = \mathrm{WPL}_n + (\mathrm{WPL}_m - \mathrm{WPL}_n - w_n) + (w_m - w_n)$, then, $\mathrm{WPL}'_n - \mathrm{WPL}_m = -2 * w_n + w_m < 0$, that is, $\mathrm{WPL}'_n < \mathrm{WPL}_m$. □

Theorem 5.2. In a storage tree with root V_m, if $w_n = \max w_i, V_i \in \mathrm{Children}(V_n)$, then, V_n is the candidate host.

Proof. Suppose V_i is one of the children nodes of the V_m, the weighted path length of V_i is WPL_i and weight is w_i. If V_i is the root of the restructured tree, then the weighted path length $\mathrm{WPL}'_i = \mathrm{WPL}_i + (\mathrm{WPL}_m - \mathrm{WPL}_i - w_i) + (w_m - w_i)$. Then, for any child node of V_m, say V_x, we can get $\mathrm{WPL}'_x - \mathrm{WPL}'_n = 2(w_n - w_x)$. Because the $w_n > w_x$, then, $\mathrm{WPL}'_x - \mathrm{WPL}'_n > 0$.

Consequently, only V_n whose weight is the biggest could be the candidate because the WPL'_n is the smallest among the current child nodes of the root. □

Theorem 5.3. For any edge e_{mn} in an optimized storage tree with root V_r, if V_n is the parent node of V_m, then $\mathrm{WPL}'_n < \mathrm{WPL}'_m$.

Proof. The V_n is the parent node of V_m, there is a path from V_r to V_m via V_n, denoted as $P = \langle V_r, V_1 \dots V_n, V_m \rangle$.

Suppose A_i is the success nodes set of $(i-1)^{\text{th}}$ node of path P except of the i^{th} node at the path P. That is, $A_i = \{v_x \mid v_x \in \mathrm{desendant}(v_{i-1}) \wedge v_x \notin \mathrm{sub}(v_i)\}$ where V_i is the i^{th} node of path P. We get $w_i = \sum_{v_x \in \mathrm{children}(v_i)} w_x$, and let $W_i = \sum_{v_x \in \mathrm{children}(v_i) \wedge v_x \in A_i} w_x$, then $W_i = w_{i-1} - w_i$.

For any e_{mn}, we can calculate the difference $\nabla_m of$ the WPL of the storage tree with V_m as the root and the WPL of the storage tree with V_r as the root.

$$\nabla_m = \mathrm{WPL}'_m - \mathrm{WPL}_r = W_1 * d + W_2 * (d-2) + \ldots + W_{d+1} * (1-d) \tag{14}$$

We also can get ∇_n,

$$\nabla_n = \mathrm{WPL}'_n - \mathrm{WPL}_r = W_1 * (d-1) + W_2 * (d-3) + \ldots + W_d * (1-d) \tag{15}$$

Thus, $\nabla_m - \nabla_n = W_1 + W_2 + \ldots + W_d - W_{d+1}$. Since V_r is the root of the storage tree, we can get $\nabla_m > 0$; thus, $\nabla_m - \nabla_n > (W_1 + W_2 + \ldots + W_d) - ((W_1 * d + \ldots + W_d * (2 - d))/d) > 0$. Then, $\nabla_m - \nabla_n = \mathrm{WPL}'_m - \mathrm{WPL}'_n > 0$. $\square$

After the weights of tree are updated, the maximum weighted child node V_s is selected for comparison with the root. If the weight of this node satisfies $(w_n - 2w_s) < 0$, then we know that the storage tree with the root of V_s is better than the original tree; otherwise the program is aborted, because the root of the optimized storage tree cannot be the node (Theorem 5.1); it is also impossible on the subtree of this node (Theorem 5.2); and other children nodes are also impossible to be the new root (Theorem 5.3). We can conclude that when the program is terminated, the new storage tree would be the optimized storage tree and the root is the new host.

6 EXPERIMENTAL ANALYSIS

In order to testify the correctness and the performance of the algorithm, we simulated 14 data servers (9 leaf nodes) and built the layered data storage system as Figure 1 shows. The communication among the data nodes is based on the TCP socket. We also implemented the data generator to simulate the generation of context data.

6.1 The Comparison of Different Hosts

Based on the layered data storage system, these two experiments show that selecting different host nodes results in different communication cost.

In the first experiment, we generate a sequence with 20 context data sending events, which is represented as the sequence of ID: 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 8, 8, 11, 11, 12, 12, 4, 4, 4, 5, 6, 7. The total number of data collections is 24 and the probability of leaf node data receiving is 0.125 (4), 0.04 (5, 6, 7), 0.08 (8, 11, 12), 0.25 (13, 14) where the number in the parentheses represents the node ID. Obviously, in this experiment, the FA-domains of the device are nodes 13 and 14.

In the second experiment, we generate the sequence as 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 13, 13, 13, 14, 14, 14, 8, 11, 11, 12, 12. The total number of data collections is still 24 while the probability is 0.17 (5, 7), 0.21 (6), 0.125 (13, 14), 0.04 (8), 0.08 (11, 12). Here, the *FA-domains* are nodes 5, 6 and 7.

Figure 3 shows the results of the communication cost of choosing different nodes as hosts. We can see that in the first experiment, if we choose node 3 or node 10 as the host node, the communication cost is minimized, which are only 50 socket communications. In the second experiment, node 2 is the best node as the host, and the number of communications is 67.

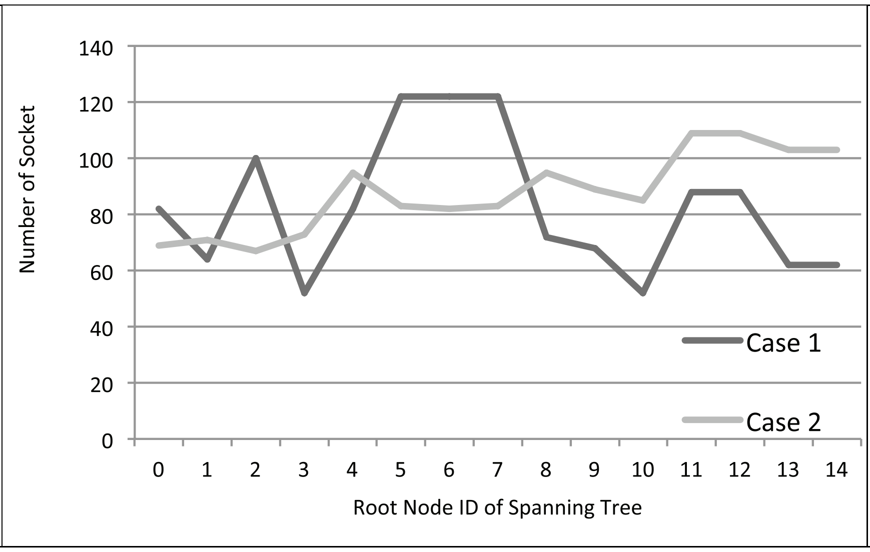


Fig. 3. Context storage tree

6.2 Performance of GHS

In Section 5, we have theoretically proved that the algorithm works correctly. In the experiment, we test two cases shown in Section 5.2, and the algorithm gives node 10 and node 2 as the result for case 1 and case 2 respectively. From Figure 4, the result is correct, except that the algorithm gives different results due to different sequence of nodes if several nodes have the same cost.

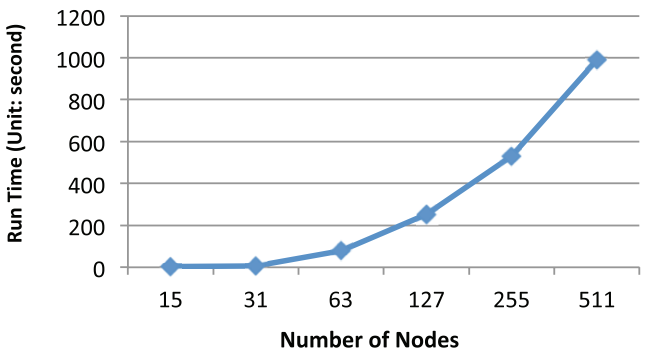


Fig. 4. Context storage tree

Secondly, we test the run time of the algorithm under conditions of different node size, supposing that the computing node has gathered all information about the whole topology. The test cases use the complete binary tree with node size of 15, 31, 63, 127, 255, 511 and the probability of each leaf node is generated by random function. The run time comparison is shown in Figure 4. The result shows that the run time of EHS algorithm grows linearly with the size of nodes when the node size is large enough.

6.3 The Performance of DHS

The goal of the DHS is to reduce the communication cost of the system during the data storage. Now we test the communication cost of DHS itself. The cost of DHS algorithm includes the cost of weight update and host reselection procedure. Figure 5 a) shows the communication cost of DHS under the situation that the device moves at a low speed, i.e. it does not frequently switch from different domains. Since most of the sensing devices move like this, we consider it as the normal case. As we can see, the cost of DHS is very low compared with the normal communication. More concretely, (Host Selection cost)/(Total cost) = 1/22. Figure 5 b) shows the communication cost when the device is moving fast, and it sends data to several different nodes in a short time. In this case, the cost of DHS occupied nearly half communication cost in some nodes, because of frequent host reselection. (Host Selection cost)/(Total cost) = 17/41.

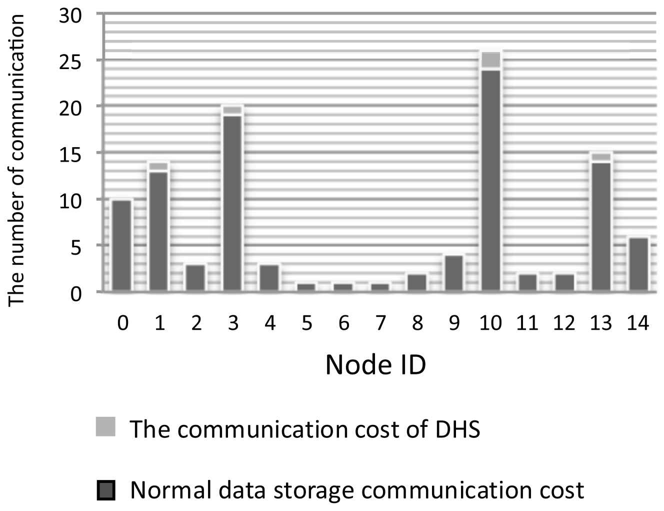
As Figure 5 b) shows, if the host was changing frequently, the cost of DHS would be high. We call the frequent host switch host jitter. In a practical system, the behaviors of sensing devices are relatively stable. The devices seldom skip between two domains and have asymmetric data sending ratio. Consequently, in a normal case, the costs of host selection only take up a small portion of the total communication costs. In some special cases, we also can set a threshold to reduce the number of host reselection, so that we can avoid the host jitter.

7 CONCLUSIONS

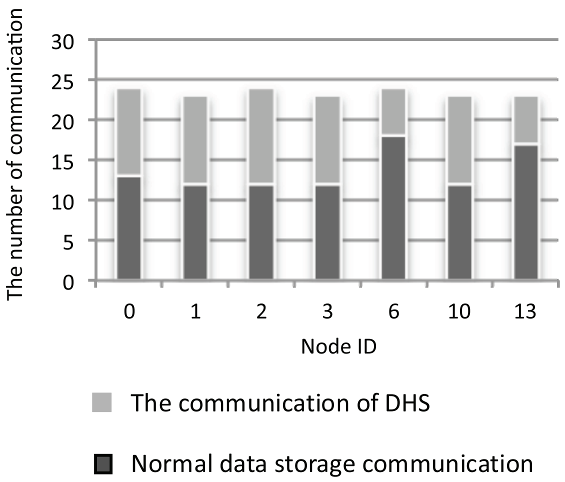
In this paper, we investigate the layered context data storage problem. The mobile devices send data to different nodes, which needs to be merged and stored in certain host nodes. How to select a suitable host with a minimum communication cost is a challenge. We propose a global host selection algorithm GHS and an on-line host selection algorithm DHS to select the host node. We also prove the correctness of two algorithms and set up simulation experiments to approve the performance of GHS and DHS. In future work, we will deploy it in a large-scale system and get more experimental data to improve the efficiency of the system.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant No. 61073028, No. 61021062, the National Basic Research Program of China (973) under Grant No. 2009CB320705.



a)



b)

Fig. 5. The cost of host selection: a) normal case, b) frequent domain switch

REFERENCES

[1] Hightower, G.—Want, R.: SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength. 2000.

[2] Zhou, G.—Liu, J.—Wan, C.—Yarvis, M.—Stankovic, J.: BodyQoS: Adaptive and Radio-Agnostic QoS for Body Sensor Networks. Proceedings of the 27th

IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, April 2008.

[3] Han, Y.—Kim, C.—Gil, J.: A Greedy Algorithm for Target Coverage Scheduling in Directional Sensor Networks. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, Vol. 1, No. 2/3, 2010.

[4] Deshpande, A.—Guestrin, C.—Madden, S.—Hellerstein, J.—Hong, W.: Model-Driven Data Acquisition in Sensor Networks. Proceedings of VLDB 2004.

[5] Madden, S.—Franklin, M.—Hellerstein, J.—Hong, W.: Tinydb: An Acqusitional Query Processing System for Sensor Networks. ACM TODS 2005.

[6] Mathur, G.—Chukiu, P.—Desnoyers, P.—Ganesan, D.—Shenoy, P.: A Storage-Centric Camera Sensor Network. Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems – Demonstrations (SenSys), Boulder CO, November 1–3, 2006.

[7] Diao, Y.—Ganesan, D.—Mathur, G.—Shenoy, P.: Rethinking Data Management for Storage-Centric Sensor Networks. Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar (CA) 2007.

[8] Qian, Z.—You, I.—Lu, Y.—Lu, S.: A Dynamic Host Selection Algorithm for Layered Data Storage Architecture in a Pervasive Space. Proceedings of Fourth International Workshop on Intelligent, Mobile and Internet Services in Ubiquitous Computing, Krakow, Poland 2010.

[9] Gonzalez, J. H. H.—Xiaolei, L.—Diego, K.: Warehousing and Analysis of Massive RFID Data Sets. Proceedings of 2006 International Conference on Data Engineering 2006.

[10] Ying Hu, S. S.—Timothy, C.—Jagannathan, S.: Supporting RFID-based Item Tracking Applications in Oracle DBMS Using a Bitmap Datatype. Proceedings of the 31st VLDB Conference 2005.

[11] Hong-Hai Do, J. A.—Hackenbroich, G.: Architecture Evaluation for Distributed Auto-ID Systems. Proceedings of International Conference on Database and Expert Systems Applications 2006.

[12] EPC global. The EPC global Architecture Framework – EPC global Final Version 1.3, approved March 19, 2009.

[13] Agrawal, R.—Cheung, A.—Kailing, K.—Schonauer, S.: Towards Traceability Across Sovereign, Distributed RFID Databases. Proceedings of the 10th International Database Engineering and Applications 2006.

[14] Chalasani, S.—Boppana, R.: Data Architecture for RFID Transactions. IEEE Transactions on Industrial Informatics 2007.

[15] UCLA WinRFID Middleware. `http://www.wireless.ucla.edu/rfid/winrfid`.

[16] Hongzi Zhu, M. L.—Zhu, Y.—Ni, L. M.: HERO: Online Real-Time Vehicle Tracking. IEEE Transactions on Parallel and Distributed Systems, Volume 20, 2009.

Zhuzhong Qian is an Associate Professor at the Department of Computer Science and Technology, Nanjing University, P. R. China. He received his Ph. D. Degree in Computer Science in 2007. Currently his research interest includes cloud computing, distributed systems and pervasive computing. He is the chief member of several national research projects on cloud computing and pervasive computing. He has published more than 30 research papers in related fields.

Youyou Lu received the Bachelor's Degree in Computer Science from Nanjing University in 2009. He is now a doctoral candidate at the Department of Computer Science and Technology, Tsinghua University. His current research interests include massive storage architecture, parallel file systems, and distributed systems.

Ilsun You received his M. Sc. and Ph. D. Degrees in Computer Science from Dankook University, Seoul, Korea in 1997 and 2002, respectively. Since March 2005, he has been an Assistant Professor in the School of Information Science at the Korean Bible University, South Korea. He is currently serving as a main organizer of international conferences and workshops such as International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), International Workshop on Mobility Management in the Networks of the Future World (MobiWorld), International Workshop on Managing Insider Security Threats (MIST), International Workshop on Security and Cognitive Informatics for Homeland Defense (SeCIHD) and so forth. He is in the Editorial Board for International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC), Computing and Informatics (CAI), and Journal of Korean Society for Internet Information (KSII). Also, he has served as a guest editor of several journals such as Information Sciences (INS), Mobile Information System (MIS), Wireless Communications and Mobile Computing (WCMC), Journal of Universal Computer Science (JUCS), Journal of Intelligent Manufacturing (JIM), Ad Hoc & Sensor Wireless Networks (AHSWN), Wireless Personal Communication (WPC), and so on. His main research interests include network security, formal security analysis and mobile internet. He is a member of the IEICE, KIISC and KSII.

Sang-Lu Lu received her B. Sc., M. S., and Ph. D. degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a Professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology, Nanjing University. Her research interests include distributed computing, pervasive computing, and wireless networks.