Sensors **2013**, *13*, 5757-5776; doi:10.3390/s130505757

OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Article

A Split-Path Schema-Based RFID Data Storage Model in Supply Chain Management

Hua Fan [1,*], Quanyuan Wu [1], Yisong Lin [2] and Jianfeng Zhang [1]

[1] School of Computer Science, National University of Defense Technology, Changsha 410073, China; E-Mails: quanyuan.wu@gmail.com (Q.W.); jfzhang@nudt.edu.cn (J.Z.)

[2] PLA General Logistics Department, Logistics Science Research Institute, Beijing 100071, China; E-Mail: linyisong@live.cn

* Author to whom correspondence should be addressed; E-Mail: huafan@nudt.edu.cn; Tel.: +86-139-7480-2264; Fax: +86-731-8457-4607.

Received: 8 March 2013; in revised form: 24 April 2013 / Accepted: 2 May 2013 / Published: 3 May 2013

Abstract: In modern supply chain management systems, Radio Frequency IDentification (RFID) technology has become an indispensable sensor technology and massive RFID data sets are expected to become commonplace. More and more space and time are needed to store and process such huge amounts of RFID data, and there is an increasing realization that the existing approaches cannot satisfy the requirements of RFID data management. In this paper, we present a split-path schema-based RFID data storage model. With a data separation mechanism, the massive RFID data produced in supply chain management systems can be stored and processed more efficiently. Then a tree structure-based path splitting approach is proposed to intelligently and automatically split the movement paths of products . Furthermore, based on the proposed new storage model, we design the relational schema to store the path information and time information of tags, and some typical query templates and SQL statements are defined. Finally, we conduct various experiments to measure the effect and performance of our model and demonstrate that it performs significantly better than the baseline approach in both the data expression and path-oriented RFID data query performance.

Keywords: RFID technology; data storage model; data compression; data processing; split-path schema; supply chain management

1. Introduction

Radio Frequency IDentification (RFID) is a kind of automatic identifying technology that allows objects, places or persons to be automatically identified at a distance without a direct line-of-sight, using an electromagnetic challenge/response exchange [1,2]. RFID has played a significant role in minimizing process costs for firms with a high value information service. With the development of low-cost passive RFID tags and vigorous RFID standardization efforts, RFID technology has become an indispensable technology in modern supply chain management [1]. The use of RFID in the supply chain management process has contributed a lot to the aspects of accuracy, information visibility and improved customer service, and supported various cost reduction factors ranging from inventory management to information and labor cost [3]. Although RFID system can provide plenty of data essential to controlling and understanding business processes, applications like supply chain management or real-time tracking may generate such a huge volume of information that it cannot be handled by traditional approaches [4]. More and more space and time are needed to store and process such huge amounts of RFID data. For example, it is predicted that only one company, such as WalMart, will generate over 7 terabytes of operational RFID data per day if it operates RFID on the item level [5]. Therefore, the storage and processing of RFID data is therefore widely considered as a principal challenge and has been an important research topic [6,7].

How can companies store and process the enormous volume of data that an RFID application will generate is a great challenge. In this paper, we present a split-path based RFID data storage model. We found that the database will have mass redundant data if we store the path of each tag independently. Therefore, the corresponding solution is proposed for the purpose of reducing the redundancy in RFID database. All the whole paths of tags have been split into two sections, and then a cluster analysis for the tags with the same path section information, including the locations and time, will be done. We also propose the corresponding approach and algorithm for splitting the whole path. Further, we design a new relational schema to store the path information and the time information for tags. The contributions of this study are as follows:

(1) Split-path based RFID data storage model. There is data redundancy in the RFID technology based supply chain management system, and there are a lot of tags with different information for the whole path but the same path information for some path sections. In the proposed model, the whole path will be split into two sections, and then be stored separately to reduce the system data redundancy.

(2) Tree structure based path splitting approach. In the supply chain, products usually have two processes successively, concentration and distribution. For this reason, there will be some positions with very high in-degree and out-degree in the supply chain. We propose a tree structure based path splitting approach, and the whole paths can be split intelligently and automatically. The large product distribution center is often the root node of the tree structure.

(3) New relational schema for RFID data storage. Based on the proposed new storage model, we redesign the relational schema to store the path information and time information of tags.

(4) Query translation. For the changes of storage model and relational schema, the original query templates have to change accordingly. Some new typical query templates are defined, and we also devise the corresponding SQL statements of these queries.

The rest of this paper is organized as follows: we discuss the related work in Section 2. Section 3 introduces the split-path based RFID data storage model and the tree structure based path splitting algorithm. In Section 4, we describe the new relational schema for RFID data storage in an RDBMS. An empirical evaluation of our solution is reported in Section 5 while our conclusions are presented in Section 6.

2. Related Work

With the development of RFID technology, more and more research on RFID data management has been done recently, such as RFID data warehousing and duplicate elimination [8–10], RFID data querying [7,11–13], RFID data cleaning [14–16], and so on. In this section, we will review the existing RFID data compression and processing approaches that are related to our work.

The special way in which RFID device gets data brings more redundancy to the RFID data sets. In recent years, several efforts have been made in the related research field. Mahdin *et al.* [8] proposed a data filtering approach that efficiently detects and removes duplicate readings from RFID data streams. However, the approach has its limitation that all filtering process is aimed at the raw data at reader level rather than the path records of tags. Gonzalez *et al.* [9] proposed a movement graph model as a compact representation of RFID data sets. It provides a clean and concise representation of large RFID data sets. This approach is based on the assumption that the products tend to move and stay together and thus can be grouped together based on their locations. However, it is useless for warehousing of products with scattered movements, and the path oriented queries is inefficient. Bashir *et al.* [10] propose an energy-efficient in-network RFID data filtering scheme to filter duplicate readings in wireless sensor network. In this schema, a clustering mechanism is used to eliminate the duplicate data, and cluster heads only need to forward filtered data towards the base station.

Bai *et al.* [11] proposed a stream query language to provide comprehensive temporal event detection, through temporal operators and extension of sliding-window constructs, and it can support the general RFID data processing for a large variety of RFID applications. Park *et al.* [12] proposed an effective technique for indexing RFID continuous queries. This technique can convert a number of segments into compressed data and store the result as one object. Furthermore, a transform technique is proposed to find a repeated group of segments and convert the group into compressed data. Wilfred *et al.* [13] presented a holistic framework that supports data querying and analysis of raw datasets obtained from different RFID collection points managed by supply chains. Lee *et al.* [7] proposed a path encoding schema to process a massive amount of RFID data for supply chain management. By using two numbers, the paths that satisfy the conditions can be found easily. However, with the increasing of the tag numbers in system, the storage cost of data and the time cost of data query will increase rapidly.

Our work on the RFID data compression and processing makes use of several traditional data processing techniques. In this paper, we propose a split-path based RFID data storage model that improves on the storage model used in Reference [7] to reduce the storage cost and speed up query processing.

3. Split-Path Based RFID Data Storage Model

In the RFID technology based supply chain management system, when the product with RFID tag moves through the detection region of the reader, it will be detected by the reader and a record will be generated in the form of (*tag_id*, *reader_id*, *timestamp*), where *tag_id* and *reader_id* refer to EPCs which universally uniquely identify the tagged item and the RFID reader (readers are usually fixed at a specific location, so *reader_id* and the locations *LOC* in supply chain are in one-to-one correspondence), and the *timestamp* is the time when the reading occurred [17]. In the work of Reference [7], the raw RFID data generated in supply chain management have been translated into the form of (*tag_id*, *loc*, *start_time*, *end_time*), which is a set of stay records and has no duplicates. *loc* is the location of the RFID reader which detects the tag; *start_time* and *end_time* are the time when the tag enters and leaves the location, respectively. Furthermore, path records are constructed to instead of raw RFID data in the form of $L_1[s_1, e_1]\rightarrow\ldots\rightarrow L_i[s_i, e_i]\rightarrow\ldots\rightarrow L_n[s_n, e_n]$, where L_i is the location where the tag is detected, s_i and e_i are the *start_time* and *end_time* at the location L_i, respectively.

Figure 1 shows the path graph of an electronics supply chain. The node A, C, D and F mean several manufacturers, and node I, J, M and N are the electronics retailers. Other nodes mean middlemen, and node O is the biggest electronics distributing center in this supply chain. The path information of products in this supply chain for a period is stored in Table 1. We found that the storage cost can be significantly reduced if we split the whole path of each product into two sections by the distributing center node O and store the information of each path section separately, and the performance of path oriented RFID data queries can also be improved. All the path records in Table 1 can be expressed by the combination of two path sections in Table 2. For example, the path information of Tag 1 can be represented by the path sections So_4 and Si_2, and all the 18 long path records in Table 1 can be represented by the 10 short path section records. The whole path graph is separated into two tree structures by the center node, and the tree structures represent the concentration and distribution of products, respectively. The topology similar to Figure 1 is very common in the RFID technology based applications, such as supply chain management, logistics management, and so on.

Figure 1. Path graph of an electronics supply chain.

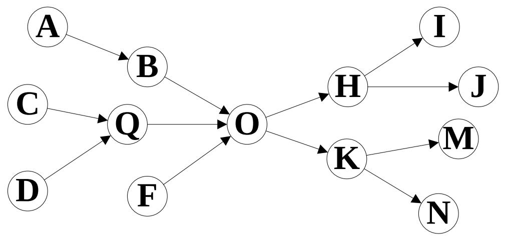


Table 1. The path records of products in supply chain for a period.

Path Records
Tag 1: A[2, 3] → B[5, 6] → O[8, 9] → H[11, 12] → I[14, 16]
Tag 2: C[3, 4] → Q[5, 7] → O[8, 9] → H[11, 12] → J[13, 15]
Tag 3: D[1, 3] → Q[5, 7] → O[8, 9] → H[11, 12] → J[13, 15]
Tag 4: A[2, 3] → B[5, 6] → O[8, 9] → H[11, 12] → J[13, 15]

Table 1. *Cont.*

Path Records
Tag 5: F[3, 6] → O[8, 9] → H[11, 12] → J[13, 15]
Tag 6: C[3, 4] → Q[5, 7] → O[8, 9] → K[10, 11] → M[12, 14]
Tag 7: D[1, 3] → Q[5, 7] → O[8, 9] → H[11, 12] → I[14, 16]
Tag 8: A[2, 3] → B[5, 6] → O[8, 9] → K[10, 11] → N[13, 15]
Tag 9: D[1, 3] → Q[5, 7] → O[8, 9] → K[10, 11] → N[13, 15]
Tag10: A[2, 3] → B[5, 6] → O[8, 9] → K[10, 11] → M[12, 14]
Tag11: C[3, 4] → Q[5, 7] → O[8, 9] → K[10, 11] → N[13, 15]
Tag12: F[3, 6] → O[8, 9] → H[11, 12] → I[14, 16]
Tag13: D[1, 3] → Q[5, 7] → O[8, 9] → K[10, 11] → M[12, 14]
Tag14: F[3, 6] → O[8, 9] → K[10, 11]→ N[13, 15]
Tag15: C[3, 4] → Q[5, 7] → O[8, 9] → H[11, 12] → I[14, 16]
Tag16: F[3, 6] → O[8, 9] → K[10, 11] → M[12, 14]
Tag17: D[1, 3] → Q[5, 7] → O[8, 9] → K[10, 11] → N[14, 16]
Tag18: C[1, 2] → Q[4, 5] → O[8, 9] → H[11, 12] → I[14, 16]

Table 2. Path section records.

Source Sections	**Sink Sections**
So_1: A[2, 3] → B[5, 6] → O[8, 9]	**Si_1:** O[8, 9] → H[11, 12] → I[14, 16]
So_2: C[3, 4] → Q[5, 7] → O[8, 9]	**Si_2:** O[8, 9] → H[11, 12] → J[13, 15]
So_3: C[1, 2] → Q[4, 5] → O[8, 9]	**Si_3**: O[8, 9] → K[10, 11] → M[12, 14]
So_4: D[1, 3] → Q[5, 7] → O[8, 9]	**Si_4**: O[8, 9] → K[10, 11] → N[13, 15]
So_5: F[3, 6] → O[8, 9]	**Si_5**: O[8, 9] → K[10, 11] → N[14, 16]

3.1. Tree Structure Based Path Splitting Algorithm

In the last section, we found that we can reduce the storage cost and improve the performance of path oriented RFID data queries by splitting the whole path of each product into two sections and storing the information of each path section in database separately. Therefore, we proposed a tree structure based path splitting algorithm in this section. First, the definitions of several important concepts are given as below:

Definition 1. Path graph $G(V, E)$ is a directed acyclic graph representing the moving path of tags. V is the set of locations, E is the set of transitions between locations. An edge $e = (v_i, v_j)$ indicates that tags can move from location v_i to location v_j.

Definition 2. The **successor node set** of node v is the set of all the successor nodes of node v in G, denoted as $Suc(v)$, while the **precursor node set** of node v is the set of all the precursor nodes of node v in G, denoted as $Pre(v)$.

Definition 3. A **whole path** p is a full movement path from a source node to a sink node in G in the form of $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_n$, where v_0 is the source node and v_n is the sink node in G. The node set and edge set of the path p are denoted as $C_V(p)$ and $C_E(p)$. The whole path set, P, is the set of all the whole paths in G. Unless specifically mentioned, the word "path" means whole path in this paper.

Definition 4. If a whole path p in G is split into two sections by a node $v_i \in C_V(p)$, then the section before v_i is called the **source section** of p and the section after v_i is called the **sink section** of p. We denote the set of source sections which end with the node v_i as $S_{v_i}^b$, and $S_{v_i}^f$ is the set of sink sections which start with the node v_i. In particular, $S_{v_i}^b = \{\tilde{v}_i\}$ when the node v_i is a source node in G, and $S_{v_i}^f = \{\tilde{v}_i\}$ when the node v_i is a sink node in G, where $\tilde{v}_i$ is a special path composed of only one node. For example, in Figure 1, S_O^b = {A→B→O, C→Q→O, D→Q→O, F→O}, S_O^f = {O→H→I, O→H→J, O→K→M, O→K→N}, $S_A^b = \{\tilde{A}\}$ and $S_I^f = \{\tilde{I}\}$.

Definition 5. For any node $v \in V$ (or edge $e \in E$), if there exists a path $p \in P$ that satisfy $v \in C_V(p)$ (or $e \in C_E(p)$), then we can say that p is the **covered path** of v (or e) in G. The **covered path set** of node v, $U_P(v)$, is defined as:

$$U_P(v) = \{ p \in P \mid v \in C_V(p)\} \tag{1}$$

Definition 6. Given $v \in V$, for any node $v_i \in V$, if there exists a path $p \in P$ that satisfy $v_i \in C_V(p)$ and $v \in C_V(p)$, then we can say that v_i is the **covered node** of v in G. The **covered node set** of node v, $U_N(v)$, is defined as:

$$U_N(v) = \{v_i \in C_V(p) \mid p \in U_P(v)\} \tag{2}$$

Definition 7. Given $v \in V$, for any edge $e_i \in E$, if there exists a path $p \in P$ that satisfy $e_i \in C_E(p)$ and $v \in C_V(p)$, then we can say that e_i is the **covered edge** of v in G. The **covered edge set** of node v, $U_E(v)$, is defined as:

$$U_E(v) = \{e_i \in C_E(p) \mid p \in U_P(v)\} \tag{3}$$

Definition 8. For any node $v_i \in U_N(v)$, if there does not exist a path $p \in P$ that satisfies $v_i \in C_V(p)$ and $v \notin C_V(p)$, then we can say that v_i is the **full-covered node** of v in G. The **full-covered node set** of node v, $\widehat{U}_N(v)$, is defined as:

$$\widehat{U}_N(v) = \{ v_i \in U_N(v) \mid U_P(v_i) \subseteq U_P(v)\} \tag{4}$$

Definition 9. For any edge $e_i \in U_E(v)$, if there does not exist a path $p \in P$ that satisfies $e_i \in C_E(p)$ and $v \notin C_V(p)$, then we can say that v_i is the **full-covered edge** of v in G. The **full-covered edge set** of node v, $\widehat{U}_E(v)$, is defined as:

$$\widehat{U}_E(v) = \{e_i \in U_E(v) \mid U_P(e_i) \subseteq U_P(v)\} \tag{5}$$

The path splitting method will influence the effect of data compression directly, but splitting paths in a path graph optimally is an NP-hard problem. Therefore, we propose a heuristic path splitting approach called tree structure based path splitting algorithm. The implementation of the proposed approach is an iterative process, and the main procedure consists of nine steps as described below:

Step 1. Compute the *section-tuples*. Compute the 4-tuples $\{d_v^b, l_v^b, d_v^f, l_v^f\}$ of each node v in G, where d_v^b and d_v^f are the size of the sets S_v^b and S_v^f, and $l_v^b$and l_v^f are the average length of the path sections in S_v^b and S_v^f, respectively. We present a simple method to compute the section-tuples of each node, and there is an iterative procedure for the computation of each element in the section-tuples.

$$d_v^b = \begin{cases} 1 & \text{if } Pre(v) = \emptyset \\ \sum_{v' \in Pre(v)} d_{v'}^b & \text{otherwise} \end{cases} \tag{6}$$

$$d_v^f = \begin{cases} 1 & \text{if } Suc(v) = \emptyset \\ \sum_{v' \in Suc(v)} d_{v'}^f & \text{otherwise} \end{cases} \tag{7}$$

$$l_v^b = \begin{cases} 0 & \text{if } Pre(v) = \emptyset \\ \frac{\sum_{v' \in Pre(v)} l_{v'}^b d_{v'}^b}{\sum_{v' \in Pre(v)} d_{v'}^b} + 1 & \text{otherwise} \end{cases} \tag{8}$$

$$l_v^f = \begin{cases} 0 & \text{if } Suc(v) = \emptyset \\ \frac{\sum_{v' \in Suc(v)} l_{v'}^f d_{v'}^f}{\sum_{v' \in Suc(v)} d_{v'}^f} + 1 & \text{otherwise} \end{cases} \tag{9}$$

Step 2. Compute the *covered-path* number. Compute the *covered-path* number of each node v (*i.e.*, the size of the *covered path set* of v, $|U_P(v)|$) which defines its *path-degree*, d_v, as:

$$d_v = |U_P(v)| = d_v^b d_v^f \tag{10}$$

Proof: $|U_P(v)|$ is the size of the covered path set of v. Let Y_v be the Cartesian product of S_v^b and S_v^f. Here, the Cartesian product is the set of the whole paths connected by any two path sections respectively from S_v^b and S_v^f via the common node v. d_v^b and d_v^f are the size of S_v^b and S_v^f respectively, thus $d_v^b d_v^f$ is the size of Y_v. For any path $p \in U_P(v)$, we can get $v \in C_V(p)$ by an application of Definition 5. The path p can be split into two sections by the node v , the source section p_1 and the sink section p_2. Applying Definition 4 we have $p_1 \in S_v^b$ and $p_2 \in S_v^f$, then $p \in Y_v$ holds. Hence, $|U_P(v)| \le d_v^b d_v^f$. Conversely, for any $p' \in Y_v$, combined by two path sections from S_v^b and S_v^f ($v \in C_V(p')$) respectively, which is a path in G ($p' \in P$) obviously, we can get $p' \in U_P(v)$ by applying Definition 5. Hence, $d_v^b d_v^f \le |U_P(v)|$. Combining the two inequalities above, we see that it suffices to require that $|U_P(v)| = d_v^b d_v^f$. This completes the proof.

Step 3. Compute the *length-difference*. Each node can split all its *covered-paths* into two sections, source sections and sink sections. The *length-difference* $\triangle_v$ of each node can be computed by

$$\triangle_v = |l_v^f - l_v^b| \tag{11}$$

Step 4. Compute the *throughput ratio*. The throughput of a specific node means the number of tags which have been in the node. Therefore, we can compute the throughput ratio, H_v, by the throughput of the current node to the sum of tags in the whole system as:

$$H_v = \frac{D_v}{\sum_{T_i}\{\exists t, \exists n \in V, location(T_i)_t = n\}} \tag{12}$$

$$D_v = \sum_{T_i}\{\exists t, location(T_i)_t = v\} \tag{13}$$

where, $location(T_i)_t$ is the location of tag T_i at time t, and D_v is the throughput of node v.

Step 5: Calculate the combined weight. Calculate the combined weight W_v for each node v, as

$$W_v = w_d d_v - w_\Delta \, \Delta_v + w_H H_v \tag{14}$$

where, w_d, w_Δ and w_H are the weighting factors for the corresponding system parameters. Note that these weighting factors can be chosen as needed such that $w_d + w_\Delta + w_H = 1$. The contribution of the individual components can be tuned by choosing the appropriate combination of the weighting factors. The first two components in the Equation (14), w_d and w_Δ, are directly related to the topology of the path graph, while the third component w_H is directly related to the real movement distribution of tags in a period. Therefore, we can get a common path splitting schema based on the topology of the path graph by increasing the values of w_d and w_Δ. In contrast, if we want to get the path splitting schema which can preferably apply to the storage of the information of the existing tags in system, we should increase the value of w_H.

Step 6. Choose the *root-node*. Choose the node with the biggest W_v as the *root-node*, denoted as R_i.

Step 7. Construct the *path trees*. With the root of R_i, construct two tree structures, the forward tree T_{if} and the backward tree T_{ib}. The construction of T_{if} : First, for each out-edge of R_i in G, add a new branch and a corresponding child node in T_{if} ; Then, do the same process to all its child nodes and grandchild nodes in T_{if} until all the leaf nodes in T_{if} do not have out-edges in G. Likewise, the backward tree, T_{ib}, can be constructed by doing the similar processes to R_i and the in-edges in G. After that, any path $p \in U_P(R_i)$ can be represented together by the path sections in T_{if} and T_{ib}.

Step 8. Remove the *full-covered sets*. Remove the nodes in $\widehat{U}_N(R_i)$ and the edges in $\widehat{U}_E(R_i)$ from G, for the covered path sets of these nodes and edges have been included in the trees T_{if} and T_{ib}. Therefore, $V = V - \widehat{U}_N(R_i)$ and $E = E - \widehat{U}_E(R_i)$.

Step 9. Repeat Steps 1–8 for the remaining nodes in G until $V = \emptyset$.

After processing by our splitting algorithm, the path graph of a complex supply chain will be split into two groups of trees, the forward trees and the backward trees. The forward tree (T_{if}) represents the path information of the tag after it reaches the root node, and its direction is from the root node to leaf nodes; the backward tree (T_{ib}) represents the path information of the tag before it reaches the root node, and its direction is from leaf nodes to the root node. All the leaf nodes in forward trees are the sink nodes in G, and all the leaf nodes in backward trees are the source nodes in G. Therefore, the path information of each tag will be split into two sections and stored into the two different trees separately.

3.2. An Illustrative Example of Path Splitting

We demonstrate our algorithm with the help of Figure 2. All the numeric values obtained from executing the path splitting procedure on the 19 nodes in Figure 2(a), are tabulated in Table 3. nto two sections, "A→C→K" and "K→L→O", and its root-node is K.

Figure 2(a) shows all the locations and paths in the path graph. The lettered nodes represent the different locations, and the directed edges between nodes signify that tags can move in the direction of the arrow. Figure 2(b,c) show the recursive process of the computation of the section-tuples in Step 1, and we use the symbol "X" to represent the values that have not be calculated yet in Figure 2(b).

Figure 2. An illustrative example of path splitting. (**a**) Topologic structure of the supply chain; (**b**) Recursive process of the computation of the section-tuples; (**c**) Section-tuples of the nodes; (**d**) Throughput and throughput ratio of nodes; (**e**) The root-node and its covered node set; (**f**) The construction of forward tree and backward tree; (**g**) Remove full-covered set. (**h**) The result of tree structure based path splitting algorithm.

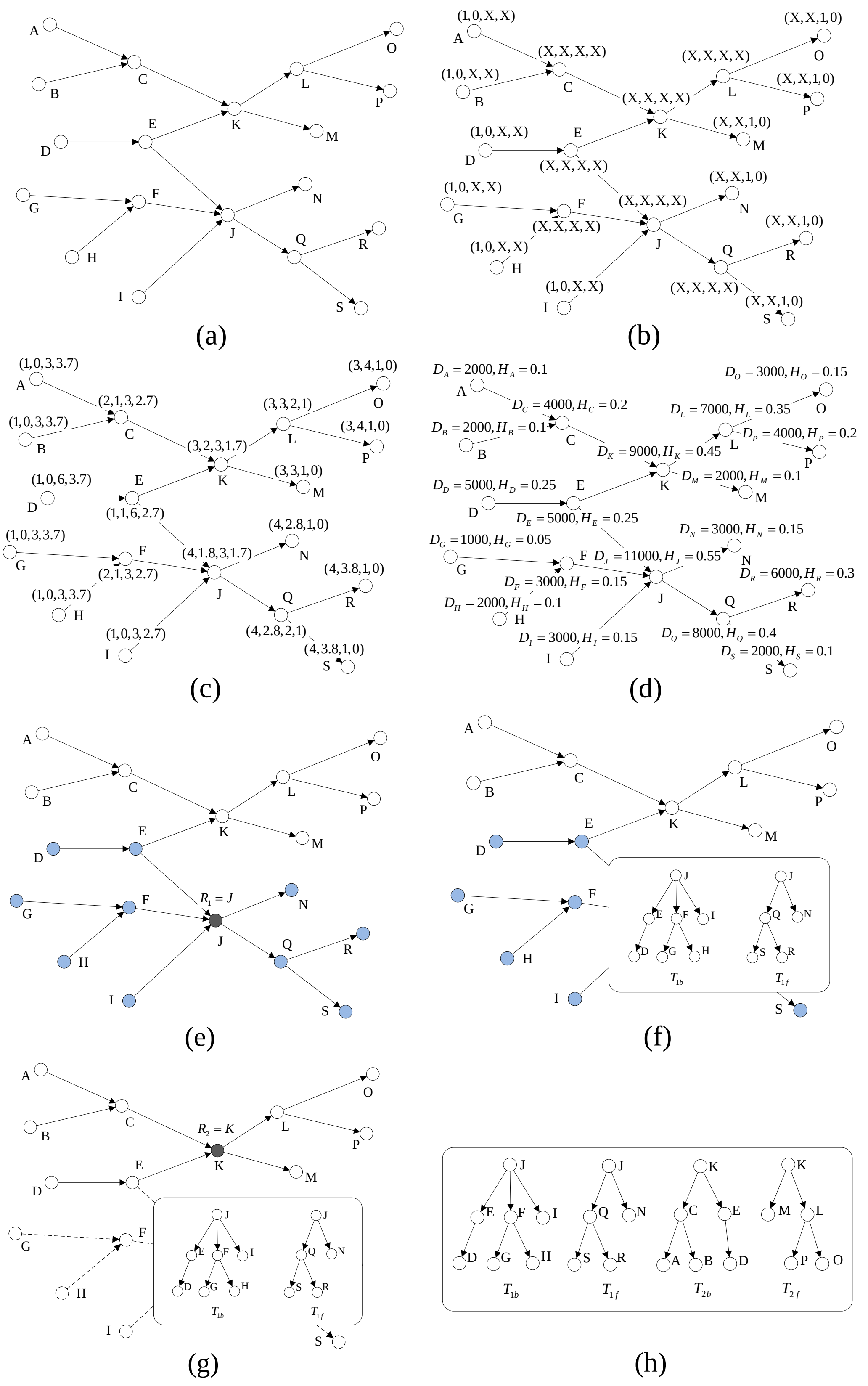


Table 3. Execution of path splitting algorithm.

Node Id	d_v^f	l_v^f	d_v^b	l_v^b	d_v	Δ_v	H_v	W_v
A	3	0	1	3.7	3	3.7	0.10	0.88
B	3	0	1	3.7	3	3.7	0.10	0.88
C	3	1	2	2.7	6	1.7	0.20	2.33
D	6	0	1	3.7	6	3.7	0.25	2.155
E	6	1	1	2.7	6	1.7	0.25	2.155
F	3	1	2	2.7	6	1.7	0.15	2.305
G	3	0	1	3.7	3	3.7	0.05	0.855
H	3	0	1	3.7	3	3.7	0.10	0.88
I	3	0	1	2.7	3	2.7	0.15	1.005
J	3	1.8	4	1.7	12	0.1	0.55	5.065
K	3	2	3	1.7	9	0.3	0.45	3.795
L	2	3	3	1	6	2	0.35	2.375
M	1	3	3	0	3	3	0.10	0.95
N	1	2.8	4	0	4	2.8	0.15	1.395
O	1	4	3	0	3	4	0.15	0.875
P	1	4	3	0	3	4	0.20	0.9
Q	2	2.8	4	1	8	1.8	0.40	3.22
R	1	3.8	4	0	4	3.8	0.30	1.37
S	1	3.8	4	0	4	3.8	0.10	1.27

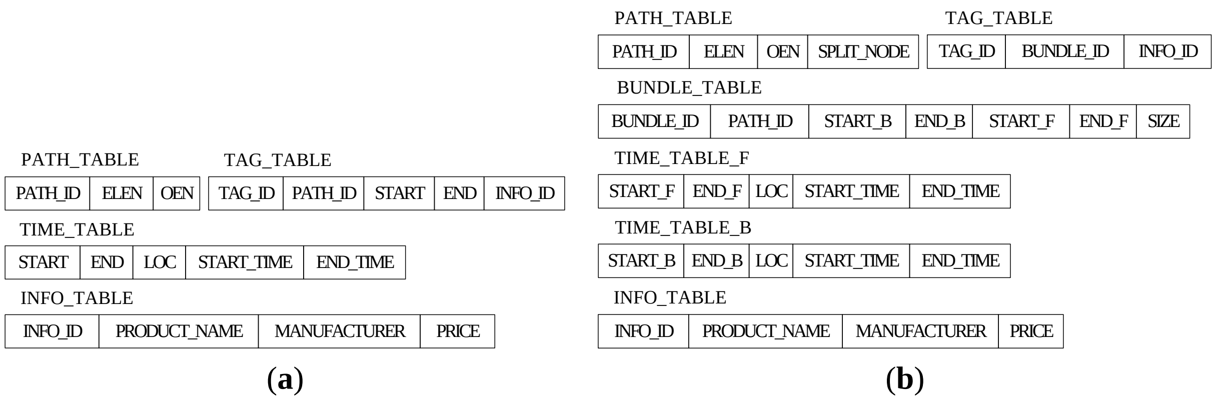
The path-degree, d_v, of each node is computed in Step 2, and the length-difference, Δ_v, of each node is calculated as Step 3. As shown in Figure 2(d), based on the value of D_v we can calculate the throughput ratio, H_v, of each node. After the values of all the components are identified, we compute the weighted metric, W_v, of each node as proposed in Step 5 of our algorithm. Here, the weighting factors considered are $\boldsymbol{w_d} = 0.4$, $\boldsymbol{w}_\Delta = 0.1$ and $\boldsymbol{w_H} = 0.5$, which is a relatively balanced choice. Figure 2(e) shows how a node with maximum W_v is selected as the root-node as stated in Step 6 of our algorithm. The solid black node represents the root-node elected for the path graph, and the blue crosshatched nodes represent the covered node set of the root-node. Figure 2(f) shows the forward tree and backward tree constructed by execution of the Step 7. As shown in Figure 2(g), the nodes in the full-covered node set and the edges in the full-covered edge set of root-node have been removed, and the solid black node is another root-node elected in the next election procedure. Figure 2(h) shows the final result, all the forward trees and backward trees constructed by our algorithm. Therefore, each whole path can be represented by two path sections, and the source section is stored in a backward tree while the sink section is stored in a forward tree. The two trees have a collective root-node which is also called the split-node of this whole path. It can be easily proved that there must be one and only one split-node for each whole path. As shown in Figure 2(h), the whole path "A→C→K→L→O" have been split into two sections, "A→C→K" and "K→L→O", and its root-node is K.

4. Relational Schema for RFID Data Storage in a RDMBS

The main aim of splitting paths is to effectively reduce the storage cost of path information and improve the performance of path oriented data queries, but which also requires us to do the corresponding adjustment to the data storage scheme and the way of data query.

Figure 3(a) shows the original relational schema to store RFID data in Reference [7]. The size of TAG_TABLE is related to the number of tags, so it is impossible to reduce the size of TAG_TABLE. However, there are many tags moving and staying together through the whole path, so we can reduce the storage cost by representing such a collective movement by a single record no matter how many tags were originally collected. For this reason, BUNDLE_TABLE has been added in our relational schema. There are seven fields in BUNDLE_TABLE. *PATH_ID* is the identifier for the path information and (*START_B*, *END_B*) and (*START_F*, *END_F*) are the identifiers for the time information of the corresponding source section and sink section. In addition, *SIZE* is the tag number of current bundle. The new TAG_TABLE in Figure 3(b) only has three fields, *TAG_ID*, *BUNDLE_ID* (the identifier for BUNDLE_TABLE) and *INFO_ID* (the identifier for INFO_TABLE).

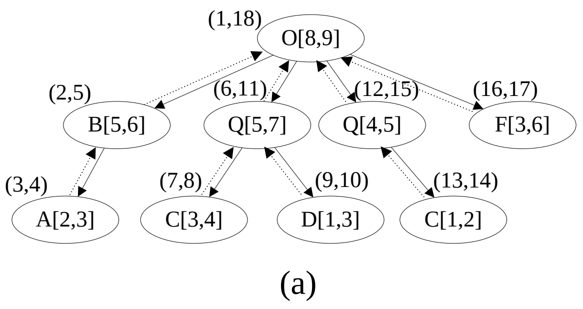
Figure 3. (**a**) Original relational schema; (**b**) New relational schema to store RFID data.

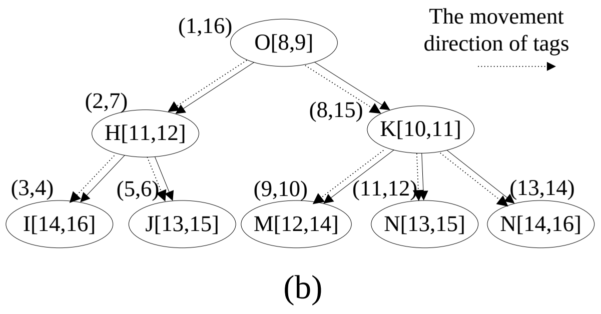

PATH_TABLE
PATH_ID | ELEN | OEN
TAG_TABLE
TAG_ID | PATH_ID | START | END | INFO_ID
TIME_TABLE
START | END | LOC | START_TIME | END_TIME
INFO_TABLE
INFO_ID | PRODUCT_NAME | MANUFACTURER | PRICE
(a)
PATH_TABLE
PATH_ID | ELEN | OEN | SPLIT_NODE
TAG_TABLE
TAG_ID | BUNDLE_ID | INFO_ID
BUNDLE_TABLE
BUNDLE_ID | PATH_ID | START_B | END_B | START_F | END_F | SIZE
TIME_TABLE_F
START_F | END_F | LOC | START_TIME | END_TIME
TIME_TABLE_B
START_B | END_B | LOC | START_TIME | END_TIME
INFO_TABLE
INFO_ID | PRODUCT_NAME | MANUFACTURER | PRICE
(b)

Obviously, the size of PATH_TABLE is related to the number of paths in the path graph, and it will be reduced if we store the split path sections instead of the whole paths. However, compared with TIME_TABLE and TAG_TABLE, the size of PATH_TABLE is much smaller. Therefore, reducing the size of PATH_TABLE has little effect on reducing the whole storage cost, and the efficiency of path oriented queries might be influenced by the complex structure of split PATH_TABLE, instead. For this reason, the structure of PATH_TABLE remains the same as before except an added label of *SPLIT_NODE*. A more detailed overview of the path encoding schema can be found in Reference [7], which we will introduce briefly in this paper. Each location in the path graph is associated with a different prime number, and the prime number for location L_a is denoted by $Prime(L_a)$. The Element List Encoding Number of the path p_i: $L_1 \rightarrow L_2 \rightarrow \ldots \rightarrow L_n$ is given by $ELEN(p_i) = Prime(L_1) \times Prime(L_2) \times \ldots \times Prime(L_n)$, and we can get the locations that compose p_i by the Element List Encoding Number $ELEN(p_i)$ based on the Fundamental Theorem of Arithmetic. The path p_i contains location L_a if and only if $ELEN(p_i) \bmod Prime(L_a) = 0$. The Order Encoding Number of p_i, $OEN(p_i)$, is the number with $0 \leq OEN(p_i) \leq ELEN(p_i)$ computed by the Chinese Remainder Theorem, and we can know the order information for any location L_a in the path by computing $OEN(p_i) \bmod Prime(L_a)$ [7]. Suppose that the locations, L_a and L_b, are the locations in the same path p_i, they have the parent-child relationship (*i.e.*, L_a/L_b) if and only if $OEN(p_i) \bmod Prime(L_a) < OEN(p_i) \bmod Prime(L_b)$ holds. Likewise, L_a and L_b have the ancestor-descendant relationship (*i.e.*, $L_a//L_b$) if and only if $OEN(p_i) \bmod Prime(L_a) + 1 = OEN(p_i) \bmod Prime(L_b)$. The symbols, “/” and “//”, are used to represent the parent-child and ancestor-descendant relationship between locations in this paper, respectively.

Except for TAG_TABLE, TIME_TABLE is the biggest table, and its size will unceasingly increase over time. Hence, reducing the size of TIME_TABLE is the key point to reduce the storage cost, and it is also one of the main emphases of this paper. Based on the forward trees and backward trees constructed in the last section, we construct the time trees to store the time information for the tags in which the node has the start time and end time as well as the location. Similarly, there are also two types of time tree, forward time tree and backward time tree. The forward time tree is responsible for storing the time information corresponding to the sink sections, and the backward time tree is responsible for storing the time information correspond to the source sections. The different nodes in the time tree represent different paths even though they have the same node id, such as C[3,4] and C[1,2] in Figure 4(a). The region-based numbering schema [18,19], which assigns a node with two values (*START_F* and *END_F* in forward time tree, or *START_B* and *END_B* in backward time tree), is used in the time tree. It encodes the starting and ending positions of a node in a path to identify the node so that the ancestor/descendant relationship between two nodes can be determined by merely examining their codes. Such a numbering schema can greatly improve the path oriented data query performance. *START_F* (*START_B*) and *END_F* (*END_B*) are assigned consecutively during the depth-first search. In the forward time tree, the region numbering has the property that node A is the ancestor of node B (A is also the precursor of B in the path) if and only if *A.START_F* < *B.START_F* and *B.END_F* < *A.END_F*. By contrast, in the backward time tree, the region numbering has the property that node A is the ancestor of node B (actually, A is the successor of B in the path) if and only if *A.START_B* < *B.START_B* and *B.END_B* < *A.END_B*.

Figure 4. Time tree. (**a**) The backward time tree; (**b**) The forward time tree.





In our relational schema, there are two time tables, TIME_TABLE_B and TIME_TABLE_F, corresponding to the backward time trees and forward time trees, respectively. The split-node (root-node) of a whole path will present in both corresponding forward time tree and backward time tree. To retrieve the time information conveniently and efficiently, we assign the region numbers that correspond to the source node (*START_B* and *END_B*) and sink node (*START_F* and *END_F*) in the path record of the tags in the specific bundle to the bundle. As shown in Figure 4, the time trees are constructed from the path records in Table 2.

We can get the time and location information for tags and bundles by their region numbers of source node and sink node. For example, if we know the region numbers of the source node and sink node of tag 11 are (7,8) and (11,12), respectively, we can retrieve the nodes satisfying *START_F* ≤ 7 and *END_F* ≥ 8 in forward time tree and *START_B* ≤ 11 and *END_B* ≥ 12 in backward time tree, such as C[3, 4], Q[5, 7], O[8, 9], K[10, 11] and N[13, 15]. We have not changed the INFO_TABLE in which the information of products such as manufacturer, price and name are stored, and the whole structure of our relation schema is shown in Figure 2(b). We take the data in Tables 1 and 2 for example to compare the changes of data storage based two different schemas. Limited by the length of paper, we only show the details of TIME_TABLE in original schema and TIME_TABLE_B and TIME_TABLE_F in our new schema. As shown in Table 4, the introduction of the split-path based RFID data storage model has greatly improved the storage efficiency and made the time information records in TIME_TABLE reduce from 41 to 17.

Table 4. Status of tables after storing trace records in Table 1. (**a**) TIME_TABLE in original schema; (**b**) TIME_TABLE_B and TIME_TABLE_F in new schema.

START	END	LOC	START_TIME	END_TIME
1	18	A	2	3
2	17	B	5	6
3	16	O	8	9
4	9	H	11	12
5	6	I	14	16
7	8	J	13	15
10	15	K	10	11
11	12	M	12	14
13	14	N	13	15
19	36	C	3	4
20	35	Q	5	7
21	34	O	8	9
22	27	K	10	11
23	24	M	12	14
25	26	N	13	15
28	33	H	11	12
29	30	I	14	16
31	32	J	13	15
37	46	C	1	2
38	45	Q	4	5
39	44	O	8	9
40	43	H	11	12
41	42	I	14	16
47	66	D	1	3
48	65	Q	5	7
49	64	O	8	9
50	55	H	11	12
51	52	I	14	16
53	54	J	13	15
56	63	K	10	11
57	58	M	12	14
59	60	N	13	15
61	62	N	14	16
67	82	F	3	6

Table 4. *Cont.*

START	END	LOC	START_TIME	END_TIME
68	81	O	8	9
69	74	H	11	12
70	71	I	14	16
72	73	J	13	15
75	80	K	10	11
76	77	M	12	14
78	79	N	13	15

(**a**) TIME_TABLE

START_B	END_B	LOC	START_TIME	END_TIME
1	18	O	8	9
2	5	B	5	6
3	4	A	2	3
6	11	Q	5	7
7	8	C	3	4
9	10	D	1	3
12	15	Q	4	5
13	14	C	1	2
16	17	F	3	6

TIME_TABLE_B

START_F	END_F	LOC	START_TIME	END_TIME
1	16	O	8	9
2	7	H	11	12
3	4	I	14	16
5	6	J	13	15
8	15	K	10	11
9	10	M	12	14
11	12	N	13	15
13	14	N	14	16

TIME_TABLE_F

(**b**)

The split-path based storage model not only can reduce the storage overhead, but also can improve the path oriented data query performances. First, the smaller sizes of the TIME_TABLE_B and TIME_TABLE_F can reduce the time cost of scanning the time information table. In addition, the adoption of the BUNDLE_TABLE enables many queries not to have to scan the TAG_TABLE, so that the execution time of queries can be reduced further.

5. Experimental Evaluation

In this section, we report our comprehensive evaluation of the proposed model and algorithms. All the experiments were conducted on an Intel(R) Core(TM) 2 Duo CPU T9550 @2.66 GHz 2.67 GHz system with 2 GB of RAM, running Windows 7. The RDBMS we used to store RFID data is Microsoft

SQL server 2005. In the experiments, we consider a comparative data size and query performance analysis of the path encoding scheme based model [7], denoted as Path, and the proposed model denoted as Split-Path.

The simulation data for our experiments were generated by a synthetic RFID data generator that simulates the operation of RFID readers in supply chain management environment. We suppose that there are 222 different positions in the whole supply chain, including two main concentration and distribution centers, 20 wholesalers, 100 manufacturers and 100 retailers. The average length of the paths is 5. There are six sets of data, which respectively include 1×10^6 tags, 2×10^6 tags, 4×10^6 tags, 6×10^6 tags, 8×10^6 tags and 1×10^7 tags, for testing the performance of our methods in the processing of RFID data with different sizes.

5.1. Query Set and Query Translation

As shown in Table 5, 9 representative queries are formulated to test various features of our model. Q1 is a tracking query, Q2–Q4 are path oriented retrieval queries, and Q5–Q9 are path oriented aggregate queries.

Table 5. Representative query set.

Query Number	Query
Q1	<TAG_ID = my_tag_id>
Q2	<//A//B/C>
Q3	<//A//B[(EndTime-StartTime) < 50]/C>
Q4	<//A//B/C, Name = 'laptop'>
Q5	<COUNT(), //A//B/C>
Q6	<AVG(B.StartTime), //A//B/C>
Q7	<AVG(C.EndTime-B.StartTime), //A//B/C>
Q8	<MIN(B.StartTime), //A//B/C>
Q9	<MIN(C.EndTime-B.StartTime), //A//B/C>

In the experiments, we store RFID data in Microsoft SQL Server, and the queries, including tracking queries and path oriented queries, must be translated into SQL queries. Because the improvement of the relational schema for RFID data storage, we have to update the query translation algorithm to get the corresponding SQL statements. We have listed some representative SQL statements in Tables 6–8, and *pA*, *pB* and *pC* in the tables below respectively denote *Prime*(*A*), *Prime*(*B*) and *Prime*(*C*):

(Q1) <TAG_ID = my_tag_id>

Table 6. SQL statements of Q1.

<TAG_ID = my_tag_id>

SELECT P.ELEN, P.OEN
FROM PATH_TABLE P, BUNDLE_TABLE B, TAG_TABLE T
WHERE T.TAG_ID= ***my_tag_id*** AND B.BUNDLE_ID=T.BUNDLE_ID AND B.PATH_ID=P.PATH_ID

(Q2) <//A//B/C>

Table 7. SQL statements of Q2.

<//A//B/C>

```
SELECT  T.TAG_ID
FROM    PATH_TABLE  P,  BUNDLE_TABLE  B,  TAG_TABLE  T
WHERE   P.ELEN%( pA*pB*pC )=0   AND   P.ELEN% pA<P.ELEN% pB   AND
        P.ELEN% pB+1=P.ELEN% pC   AND   B.PATH_ID=P.PATH_ID   AND
        T.BUNDLE_ID=B.BUNDLE_ID
```

(Q3) <//A//B[(EndTime-StartTime) < 50]/C>

Table 8. SQL statements of Q3.

<//A//B[(EndTime-StartTime)<50]/C>

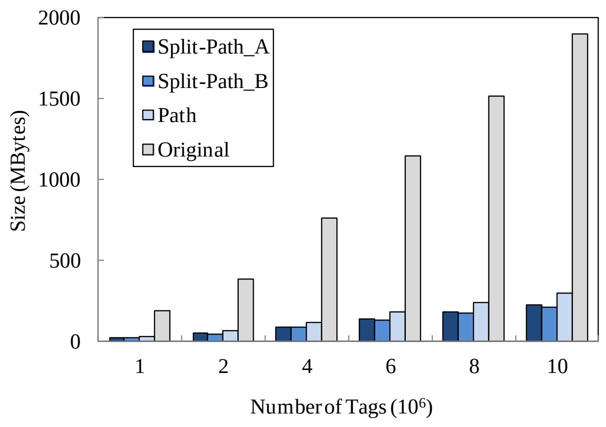
```
SELECT   T.TAG_ID
FROM     PATH_TABLE  P,  BUNDLE_TABLE  B,  TAG_TABLE  T,  TIME_TABLE_B  TB
WHERE    P.ELEN%( pA*pB*pC )=0   AND   P.ELEN% pA<P.ELEN% pB   AND
         P.ELEN% pB+1=P.ELEN% pC   AND   B.PATH_ID=P.PATH_ID   AND
         T.BUNDLE_ID=B.BUNDLE_ID   AND   TB.LOC='B'   AND
         TB.START_B<=B.START_B   AND   TB.END_B>=B.END_B   AND
         TB.END_TIME-TB.START_TIME<50
UNION
SELECT   T.TAG_ID
FROM     PATH_TABLE  P,  BUNDLE_TABLE  B,  TAG_TABLE  T,  TIME_TABLE_B  TF
WHERE    P.ELEN%( pA*pB*pC )=0   AND   P.ELEN% pA<P.ELEN% pB   AND
         P.ELEN% pB+1=P.ELEN% pC   AND   B.PATH_ID=P.PATH_ID   AND
         T.BUNDLE_ID=B.BUNDLE_ID   AND   TF.LOC='B'   AND
         TF.START_B<=B.START_B   AND TF.END_B>=B.END_B   AND
         TF.END_TIME-TF.START_TIME<50
```

Limited by the length of paper, the rest SQL statements are not listed here.

5.2. Data Compression

In Microsoft SQL server, the data is stored in an mdf file. As shown in Figure 5, we compare the storage cost of the proposed model denoted as Split-Path_A(w_d = 0.4, w_Δ = 0.1 and w_H = 0.5) and Split-Path_B(w_d = 0.3, w_Δ = 0.05 and w_H = 0.65), the path encoding schema based storage model denoted as Path, and the original raw RFID data denoted as Original. In this experiment, we can clearly see that the storage cost of Split-Path_A and Split-Path_B are always smaller than that of Path. As a matter of fact, the time information storage cost of our model (the total size of TIME_TABLE_B and TIME_TABLE_F) is only 4% of that of the path encoding schema based storage model (the size of TIME_TABLE). It is worthwhile to note that the proposed model can achieve higher compression ratio with the increasing of the tag number. Therefore, the larger the original data size of the system is, the better the effects of data expression is. In addition, we can see that the proposed model with higher value of w_H can preferably apply to the storage of the information of the existing tags in system.

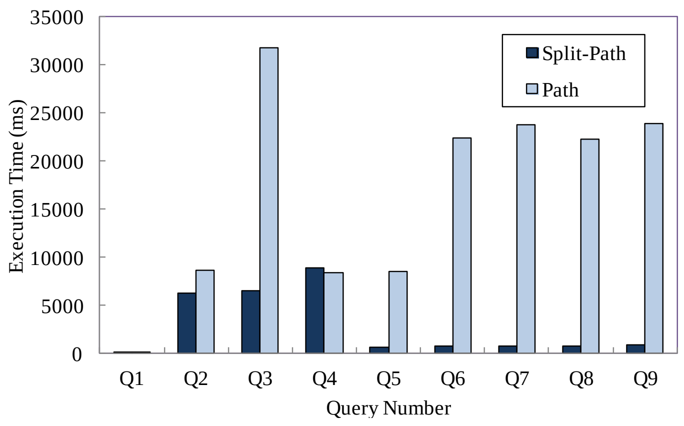
Figure 5. Storage cost comparison under different storage schema.



5.3. Query Processing

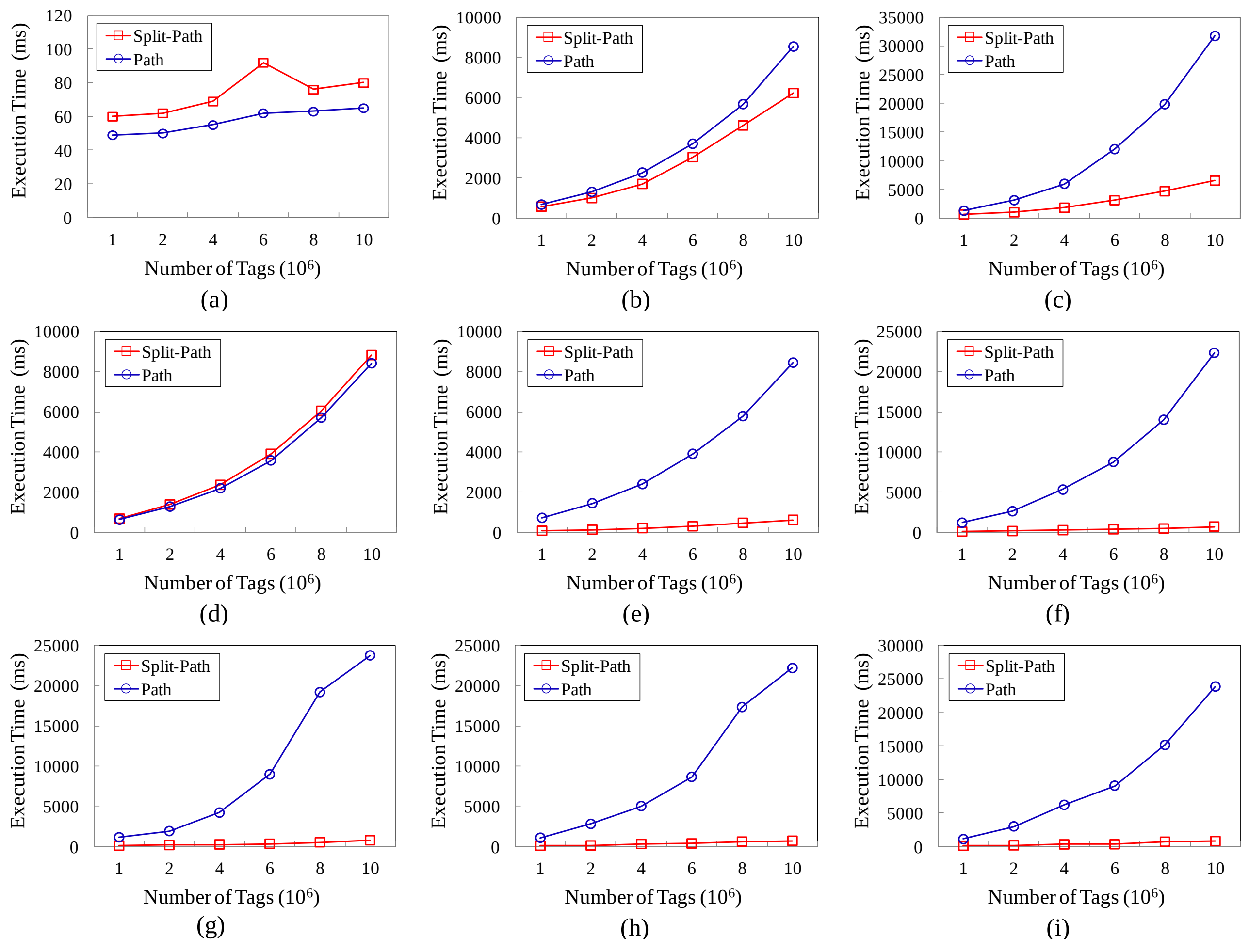
We conduct experimental evaluations for the nine representative queries in Table 5 to validate our approach in this section. In this experiment, the number of tag is 1×10^7, and we compare the query performance of two models under this condition. Figure 6 presents the average execution time of various queries. We can see that the split-path based storage model can achieve better query performance than the path encoding schema based storage model, especially for the path oriented aggregate queries.

Figure 6. Query performance comparison.



We compare the query performance of the two models according to the number of tags, and the results are shown in Figure 7. As shown in Figure 7(a,b,d), the query performances of the two models for Q1, Q2 and Q4 are very close to each other. However, as shown in the rest part of Figure 7, the performances of our approach are obviously better than that of the path encoding schema based storage model, especially for the path oriented aggregate queries. The better query performances of our model benefit from the improvement in two aspects. On the one hand, the smaller sizes of the TIME_TABLE_B and TIME_TABLE_F can reduce the time cost of scanning the time information table; on the other hand, the adoption of the BUNDLE_TABLE enables many queries not to have to scan the TAG_TABLE, so that the execution time have been further reduced.

Figure 7. Execution time for representative queries. (**a**) Query 1; (**b**) Query 2; (**c**) Query 3; (**d**) Query 4; (**e**) Query 5; (**f**) Query 6; (**g**) Query 7; (**h**) Query 8; (**i**) Query 9.



6. Conclusions

In this paper, we present a split-path based RFID data storage model to reduce the time and space overhead of RFID data processing in supply chain management systems. We split all the path records of products into two sections, and the information of these path sections is stored in database separately. Because splitting paths in a supply chain optimally is an NP-hard problem, a heuristic tree structure based path splitting approach is proposed to split the paths intelligently and automatically. In addition, based on the proposed storage model, we design a new relational schema to store the path information and time information of tags, and some typical query templates and the corresponding SQL statements is defined. Finally, the experimental results demonstrate that the proposed model and algorithm provide superior query performance and offer a significant improvement in data compression compared to the baseline approaches.

Acknowledgments

This research is supported by the National High Technology Research and Development Program of China (No. 2010AA012505, No. 2011AA010702, No. 2012AA01A401 and No. 2012AA01A402), the National Natural Science Foundation of China (No. 60933005, No. 91124002), the Support

Science and Technology Project of China (No. 2012BAH38B04, No. 2012BAH38B06) and the Information Safety Plan of China 242 (No. 2011A010).

Conflict of Interest

The authors declare no conflict of interest.

References

1. Roy, W. The magic of RFID. *ACM Queue* **2004**, *2*, 40–48.
2. Evan, W.; Leilani, B.; Garret, C.; Kayla, G.; Kyle, R.; Samuel, R.; Magdalena, B.; Gaetano, B. Building the internet of things using RFID: The RFID ecosystem experience. *IEEE Int. Comput.* **2009**, *13*, 48–55.
3. Dash, D.P. Supply chain management: The RFID advantage. *IUP J. Supply Chain Manag.* **2011**, *8*, 42–57.
4. Trujillo, R.R. Privacy in RFID and Mobile Objects. Ph.D. Theses, University Rovira I Virgili, Tarragona, Spain, 21 June 2012.
5. Gonzalez, H.; Jiawei, H.; Xiaolei, L.; Klabjan, D. Warehousing and Analyzing Massive RFID Data Sets. In Proceedings of the 22nd International Conference on Data Engineering, Atlanta, GA, USA, 3–7 April 2006; pp. 83–92.
6. Darcy, P.; Pupunwiwat, P.; Stantic, B. The Challenges and Issues Facing the Deployment of RFID Technology. In *Deploying RFID—Challenges, Solutions, and Open Issues*, 1st ed.; Turcu, C., Eds.; InTech: Rijeka, Croatia, 2011; pp. 1–28.
7. Chun-Hee, L.; Chin-Wan, C. RFID data processing in supply chain management using a path encoding scheme. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 742–758.
8. Mahdin, H.; Abawajy, J. An Approach to Filtering Duplicate RFID Data Streams. In *U- and E-Service, Science and Technology*; Kim, T.-H., Ma, J., Fang, W.-C., Park, B., Kang, B.-H., Ślęzak, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 124, pp. 125–133.
9. Gonzalez, H.; Jiawei, H.; Hong, C.; Xiaolei, L.; Klabjan, D.; Tianyi, W. Modeling massive RFID data sets: A gateway-based movement graph approach. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 90–104.
10. Bashir, A.K.; Lim, S.-J.; Hussain, C.S.; Park, M.-S. Energy efficient in-network RFID data filtering scheme in wireless sensor networks. *Sensors* **2011**, *11*, 7004–7021.
11. Yijian, B.; Fusheng, W.; Peiya, L.; Zaniolo, C.; Shaorong, L. RFID Data Processing with a Data Stream Query Language, In Proceedings of the 23rd International Conference on Data Engineering, Istanbul, Turkey, 15–20 April 2007; pp. 1184–1193.
12. Park, J.; Hong, B.; Ban, C. An efficient query index on RFID streaming data. *J. Inf. Sci. Eng.* **2009**, *25*, 921–935.
13. Ng, W. Developing RFID Database Models for Analysing Moving Tags in Supply Chain Management. In Proceedings of the 30th International Conference on Conceptual Modeling, Brussels, Belgium, 31 October–3 November 2011; pp. 204–218.
14. Fan, H.; Wu, Q.; Lin, Y. Behavior-based cleaning for unreliable RFID data sets. *Sensors* **2012**, *12*, 10196–10207.

15. Shawn, R.J.; Minos, G.; Michael, J.F. Adaptive Cleaning for RFID Data Streams. In Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB Endowment, Seoul, Korea, 12–15 September 2006; pp. 163–174.
16. Haiquan, C.; Wei-Shinn, K.; Haixun, W.; Min-Te, S. Leveraging Spatio-Temporal Redundancy for RFID Data Cleansing. In Proceedings of the 2010 International Conference on Management of Data, Indianapolis, IN, USA, 6–11 June 2010; pp. 51–62.
17. EPC Global. Available online: http://www.gs1.org (accessed on 20 February 2013).
18. Zhang, C.; Naughton, J.; DeWitt, D.; Luo, Q.; Lohman, G. On supporting containment queries in relational database management systems. *SIGMOD Rec*. **2001**, *30*, 425–436.
19. Yoshikawa, M.; Amagasa, T.; Shimura, T.; Shunsuke, XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Trans. Internet Technol.* **2001**, *1*, 110–141.