

Short Paper

An Efficient Query Index on RFID Streaming Data*

JAEKWAN PARK¹, BONGHEE HONG² AND CHAEHOON BAN³

¹*Information and Technology Laboratory
LGE Advanced Research Institute
Seoul, 137-130 Korea*

²*Department of Computer Engineering
Pusan National University
Busan, 609-735 Korea*

³*Department of Internet Business
Kosin University
Busan, 606-701 Korea*

This study introduces an efficient query indexing method for processing data stream in *RFID* system. Several approaches to build an index on queries rather than data records, called a *query index*, have been proposed and are widely used to evaluate continuous queries over streaming data. However, it is difficult to use them for the *RFID* system because they suffer from data composed of a large number of segments. To solve this problem, our study defines concepts and properties of data in the *RFID* system and then proposes a transformation method that converts a group of segments into compressed data. This allows query index to store one object instead of the segments. We measure the performance improvement of the index by the proposed technique using experimental evaluation.

Keywords: RFID system, middleware, continuous query, stabbing query, query index

1. INTRODUCTION

With the development of *RFID* (Radio Frequency Identification) technology, more and more *RFID* applications have been deployed. There are various components of a typical *RFID* deployment, namely application, middleware and edge software, within and also across corporate boundaries. Among these, the middleware is a media layer between *RFID* physical equipment and back-end application. It implements real-time data collection and *RFID* device management, and also integrates with back-end application system through the pre-defined application logic. Availability of middleware will help the *RFID* implementer to write software concentrating only on the business aspects of the enterprise application.

Recently, Major companies (such as Sun, IBM, Oracle) introduce *RFID* middleware which process *RFID* tag data cause of extending *RFID* related technology and applica-

Received July 20, 2007; revised December 10, 2007; accepted March 13, 2008.

Communicated by Suh-Yin Lee.

* Part of this paper has been presented in the 13th IEEE International Conference on Embedded and Real-time Computing Systems and Applications, August 21-24, 2007, Daegu, South Korea.

tion. *RFID* middleware collects and filters the data acquired from the readers to process application's requests. These requests are called continuous queries because they are executed continuously during tag movement. The middleware systems must build an index to process these continuous queries efficiently. The approach that builds an index on queries rather than data records, called *query index*, is suited for *RFID* middleware systems because there are fewer queries than data records and the queries are active during certain periods.

Several methods related to the query index have been proposed. The *CQI* [4] and *VCR* [8] methods build an index on continuous range queries over the location data stream of moving objects. These methods divide the continuous query using fixed-size rectangles and insert the rectangles into the query index. The search in these methods is to find the rectangles containing the positions of the moving objects. Sensor data stream systems, such as NiagaraCQ [11] and TelegraphCQ [12], adopt a query indexing scheme. They use a predicate index that is similar to the IBS-tree [13], which is a binary search tree that is created for each attribute. This method stores the ranges of an attribute value that makes a query predicate true. The search in this method finds continuous queries whose predicates are satisfied by a value sensed on a sensor node.

EPCglobal, which is a standard association devoted to *RFID* systems, has established the *ECSpec* (Event Cycle Specification) [1], which is a standard query interface for *RFID* applications. There are predicates as parameters of the interface for setting up various queries. The predicates are composed of filtering conditions of *RFID* readers and tags. As an example of an *ECSpec*, consider "readerID = 1 ~ 3, EPC pattern = <10. [1-2]. [3001-4000]>". The first predicate is a range of identifiers of readers and the second predicate is an *EPC* (Electronic Product Code) pattern of tags. Note that the *EPC* pattern is represented as a set of ranges. To follow the standard, the *RFID* query index should be built on continuous queries representing the predicates of *ECSpec* and the index should support queries, called *stabbing queries* [17], which occur when an *RFID* reader identifies each tag individually. In this case, the continuous query is represented as a number of segments in two-dimensional space composed from the Reader Identification Domain (*RID*) and the Tag Identification Domain (*TID*), because it has a range on *RID* and a set of ranges on *TID*.

In the previous studies, the query indexes have treated single data, such as a region or an interval. However, the data in an *RFID* query index is represented as a number of segments, because the data is a continuous query from the *ECSpec*. The problem with using any of the existing query index schemes for such data is that it takes a long time to build the index and to evaluate stabbing queries. It is very time consuming to build an index on such data because it is necessary to insert huge segments into the index to store a continuous query from the application. It is also inefficient to process stabbing queries that find data in the query index, because of the large size of the index after many insertions. In the paper [20], the problem was mentioned briefly but the analysis and description for the solution was so weak due to the paper length. Extending the initial paper, this study gives technical analysis more deeply, describes proposed methods in detail. In addition to the extension, this study modifies some assertions and includes several new formulas resulting in additional research outcome.

In this paper, we propose an effective technique for indexing *RFID* continuous queries. The basic idea is to convert a number of segments into compressed data and to store

the result as one object. To do this, we analyze the continuous queries and define their congruent relationship and regular repetition. Then, we propose a transform technique, called *aggregate transformation*, which finds a repeated group of segments and converts the group into compressed data. That is, this compressed data is the transformed representation of the repeated segments.

The paper is organized as follows. Section 2 describes the problem of indexing *RFID* continuous queries. Section 3 proposes an effective technique for indexing *RFID* continuous queries. Section 4 shows the superiority of the proposed method over earlier indexes. Finally, section 5 summarizes the paper.

2. INDEXING PROBLEM OF CONTINUOUS QUERIES

*EPC*global proposed the *ECSpec*, the standard interface for collecting and filtering *RFID* streaming data. The *ECSpec* defines the filtering conditions of readers and tags. The condition of readers is represented as a range of identifiers. The condition of tags is represented as an *EPC pattern* [1], which is a representation that specifies the ranges of manager, product and serial codes individually. In the *EPC* pattern $\langle A.B.C \rangle$, A means the value of the manager, B means the value of the product and C means the serial number of the item. In [1], each part of the pattern can be one of *constant*, *[low-high]* or ***. The *constant*, *[low-high]* and *** represent a value, a range and any value respectively. Fig. 1 shows examples of *EPC* patterns[†]. *EPC_Pattern*₁ ($\langle 0. 0. 1 \rangle$) is 1 and *EPC_Pattern*₂ ($\langle 0. 1. 0 \rangle$) is 2^{36} . *EPC_Pattern*₃ ($\langle 0. 1. [3-4] \rangle$) is a range $\{2^{36} + 3 \leq tid \leq 2^{36} + 4\}$ because the pattern means $\{ \langle 0. 1. 3 \rangle \cup \langle 0. 1. 4 \rangle \}$. Finally, *EPC_Pattern*₄ ($\langle 0. [3-4]. [3-4] \rangle$) consists of two ranges $\{3 \cdot 2^{36} + 3 \leq tid \leq 3 \cdot 2^{36} + 4\}$ and $\{4 \cdot 2^{36} + 3 \leq tid \leq 4 \cdot 2^{36} + 4\}$ because the pattern means $\{ \langle 0. 3. [3-4] \rangle \cup \langle 0. 4. [3-4] \rangle \}$. That is, the *EPC* pattern is represented as a set of ranges on the *TID*.

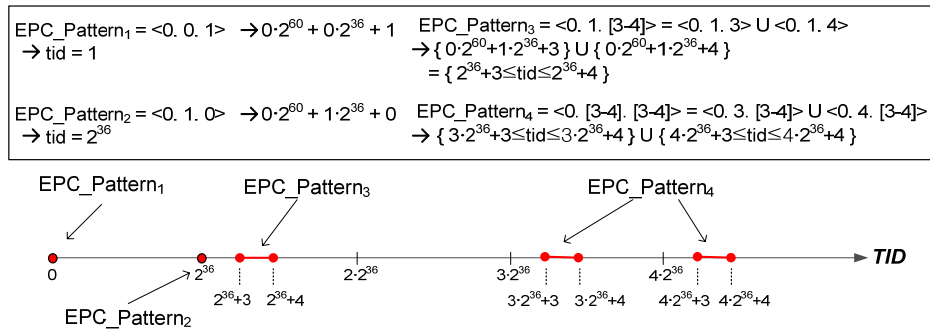


Fig. 1. An example of *EPC* patterns.

Applications register various queries to the *RFID* middleware. Middleware maintains a query index to process queries efficiently from applications. Continuous query from the application becomes data in the query index. The data, which we call *query data*, is an object that has a range on the *RID* axis and a set of ranges on the *TID* axis as

[†] We assume that the EPC for tag is a 96-bit EPC, General Identifier (GID-96) in Tag Data Standard [2]. The EPC is composed of three parts – *Industry* (28bits), *Product* (24bits) and *Serial* (36bits).

mentioned above. On the other hand, tag data identified at *RFID* reader becomes query in the query index. We call this *stabbing query* which finds query data in the index. Hence, the general stabbing query is represented as $\{(rid_i, tid_j) \in R^2 \text{ where } rid_i \in RID, tid_j \in TID\}$.

The query data in two-dimensional space (*RID*, *TID*) are complex objects composed of discrete segments. For example, as shown in Fig. 2, if a user searches for information about “in warehouse A, items that are mobile phones from SAMSUNG Electronics made this year” then the application sends an *ECSpec* representing a user query: $readerID = 1$, $EPC_Pattern = \langle 10 \cdot [1-3], [3001-4000] \rangle$, assuming that the *ID* of the reader installed in warehouse A is 1. The user query arrives at the *RFID* middleware and is inserted into the query index as data. The query data has three discrete segments. The reason is that the *EPC* pattern specifies the discrete ranges on *TID*. The query data consists of many segments if the product and serial values of the *EPC* pattern are ranges. According to the *TDS* [2], query data can have maximum 2^{24} segments. For this reason, it is much time consuming to build an index on these data because it is necessary to insert many segments into the index to store a continuous query of application. It is also inefficient to process the stabbing queries because the size of the index is very large after many insertions. To avoid many insertions, multi-dimensional indexes, such as *R-tree* [19], can be used to these query data. In this case, the index must store data in the space of four dimensions: *RID*, manager, product and serial. As we know, the query performance of the index deteriorates exponentially with higher dimensions [18]. Thus, the traditional indexes can avoid many insertions of data, but they are not efficient to process queries.

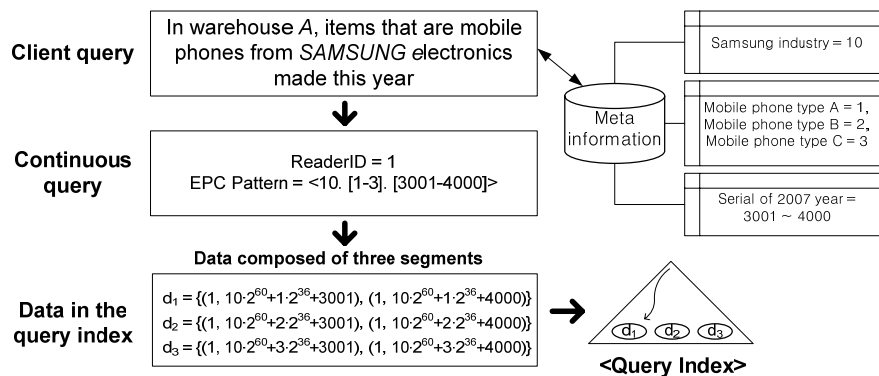


Fig. 2. An example of query data.

3. AN EFFICIENT TRANSFORM SCHEME

In the literature, several works are proposed to transform objects into different representations. There have been two approaches [3] for transformation of object representation. The first is to transform an object into a higher dimensional point and the second is to transform an object into a set of one-dimensional intervals. Unlike these methods, we propose a technique to transform the query data composed of multiple segments into a simple object.

Query data in the query index are dependent upon the *EPC* pattern. Thus, it is nec-

essary to study all the *EPC* patterns to discover properties of them. We have analyzed 27 patterns because each part of the pattern is a constant, [low–high] or *. As shown in Table 1, 11 cases are meaningful patterns among the 27 cases of syntactic combination. The first part of the *EPC* pattern identifies the manager, the second identifies the product and the third identifies the serial. The purpose of the manager is to give a unique global number to each industry, while the product and serial are numbered according to local rules of each industry. Therefore, there are illegal cases in which the manager part is a range in the *EPC* pattern with the exception of $\langle [a_1 - a_2]. *. * \rangle$ and $\langle *. *. * \rangle$. The results of the case study show that the query data consist of single or multiple segments and the number of segments in *multiple* cases is related to the value of the product part. Query data consists of single or multiple segments in two-dimensional space (*RID*, *TID*). We call query data with only one segment *simple query data*. We also call query data that is composed of multiple segments *complex query data*. Complex query data has two or more segments, up to a maximum 2^{24} segments. A segment, called a *segment of query data*, is the smallest unit organizing the complex query data. Assuming that query data is d , then we get $d = \{d_1, \dots, d_n\}$ where $1 \leq n \leq 2^{24}$. A segment of query data is represented as $d_i = \{(\min_{rid}, \min_{tid}), (\max_{rid}, \max_{tid})\}$ where $d_i \in d$.

Table 1. The results of a case study of EPC patterns.

<i>EPC</i> Patterns ($a. x. x$)	<i>EPC</i> Patterns ($[a_1 - a_2]. x. x$)	<i>EPC</i> Patterns ($*. x. x$)
$a. b. c$	$[a_1 - a_2]. b. c$	$*. b. c$
$a. b. *$	$[a_1 - a_2]. b. *$	$*. b. *$
$a. b. [c_1 - c_2]$	$[a_1 - a_2]. b. [c_1 - c_2]$	$*. b. [c_1 - c_2]$
$a. *. c$	$[a_1 - a_2]. *. c$	$*. *. c$
$a. *. *$	$[a_1 - a_2]. *. *$	$*. *. *$
$a. *. [c_1 - c_2]$	$[a_1 - a_2]. *. [c_1 - c_2]$	$*. *. [c_1 - c_2]$
$a. [b_1 - b_2]. c$	$[a_1 - a_2]. [b_1 - b_2]. c$	$*. [b_1 - b_2]. c$
$a. [b_1 - b_2]. *$	$[a_1 - a_2]. [b_1 - b_2]. *$	$*. [b_1 - b_2]. *$
$a. [b_1 - b_2]. [c_1 - c_2]$	$[a_1 - a_2]. [b_1 - b_2]. [c_1 - c_2]$	$*. [b_1 - b_2]. [c_1 - c_2]$

In the results of analysis, we have found two properties of query data. The first is that segments of complex query data are related to each other geometrically. The relationship between the segments is *congruence*: their shape and size are the same but they are located at different positions in the geometry. The general form of the *EPC* pattern in complex query data is $\langle M_1. [P_1 - P_2]. [S_1 - S_2] \rangle$. Assume that complex query data is composed of a range (rid_a, rid_b) on the *RID* axis and an *EPC* pattern $\langle m_1. [p_1 - p_2]. [s_1 - s_2] \rangle$ on the *TID* axis and the data d is composed of the segments $\{d_0, d_1, \dots, d_{p_2-p_1}\}$. Then, a congruence relationship (denoted by \equiv) exists between d_i and d_j where $0 \leq i < j \leq p_2 - p_1$. Another property of complex query data is that the segments are located with regular gaps along the *TID* axis. Each segment exists in the same *RID* and successive distances between the starting point of the segment d_i and the starting point of the segment d_{i+1} in *TID* is 2^{36} .

Complex query data are composed of many segments, but they have properties of congruence and regular repetition between segments of them. We convert the complex query data into a simplified form based on the properties. To do this, it is important to

find the repeated form of the data. A regular grid structure is a good conceptual tool to extract a regularly repeated shape of the data, because it is composed of fixed cells that are repeated regularly, like the complex query data. We propose a new transform technique, called aggregate transformation, using a virtual grid structure. This transform method consists of three steps. The first step is to extract a repeated form of query data using the virtual grid, the second step is to calculate a range of grid cells covering the data and the final step is to aggregate the repeated form and the range in transformed space. To explain the steps in details, we define the *repeated pattern (RP)* of query data which means the shape in the cells when query data is overlaid with grid cells in two-dimensional space. $RP = \{((\min_{rid}, \min_{tid}), (\max_{rid}, \max_{tid})) \subset R^2\}$.

The first step of the transformation is to extract the *RP* from complex query data. To describe the steps in details, we consider following assumptions:

- Let query data D_1 composed of a range $\overline{(rid_a, rid_b)}$ on the *RID* axis and *EPC* pattern $\langle m_1, [p_1 - p_2], [s_1 - s_2] \rangle$ on the *TID* axis;
- Let d_0, d_1, \dots , and $d_{p_2-p_1}$ be the segments of data D_1 ;
- Let h_1 and h_2 be the shortest and longest distance on the *TID* axis between a segment d_i and the bottom-left point of a cell containing the bottom-left point of d_i .

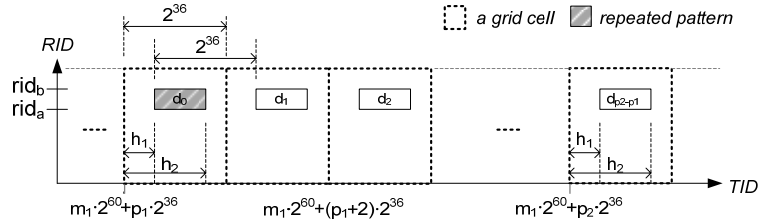
The mechanisms to extract the *RP* are classified into two cases according to the size of the grid cell. Fig. 3 (a) shows the first case: the length of a grid cell on the *TID* axis is equal to or smaller than the gap between segments of query data. We assume that the length of a grid cell on the *TID* axis is 2^{36} . In this case, the segments d_0, d_1, \dots , and $d_{p_2-p_1}$ are the same shape in the space of a cell that contains each segment according to the properties (congruence relationship, repeated repetition) of query data. This shape is the repeated pattern of the data. According to the definition, the *RP* is $\{(rid_a, h_1), (rid_b, h_2)\}$. Fig. 3 (b) shows the second case: the length of a grid cell on the *TID* axis is larger than the gap between segments of query data. We assume that the length of a grid cell on the *TID* axis is 2^{37} . In this case, two segments are within a cell and a group of them is repeated regularly. Thus, the group becomes *RP* and it is represented as $\{(rid_a, h_1), (rid_b, 2^{36} + h_2)\}$. Generally, if the length (2^K) of a cell on the *TID* axis is larger than the gap (2^{36}) between segments of query data, then the number of segments within a cell becomes 2^{K-36} .

Regardless of the two cases mentioned above, the segments of query data are split by the grid if each of them is overlapped with more than two cells because the *RP* is a shape within a cell according to the definition of the *RP*. Fig. 3 (c) shows the splitting example. We assume that the length of a grid cell on the *TID* axis is 2^{35} , and the distance values are $h_1 < 2^{35} < h_2$. In this case, every segment is overlapped with two cells as shown in Fig. 4. According to the definition of the *RP*, the query data is split into two *RPs*. The first RP_1 becomes $\{(rid_a, h_1), (rid_b, 2^{35})\}$ and the second RP_2 becomes $\{(rid_a, 0), (rid_b, h_2 - 2^{35})\}$. However, splits do not occur when the length of a cell on the *TID* axis is equal to or larger than 2^{36} , because the maximum length of a segment is smaller than 2^{36} on the *TID* axis.

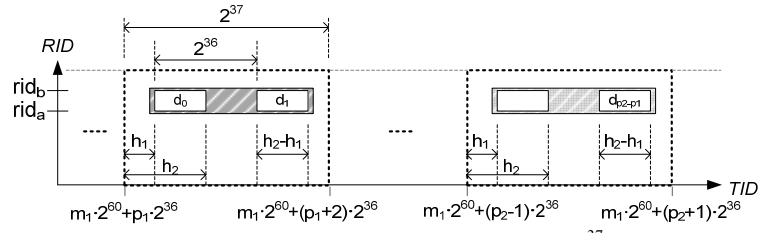
The second step of the transformation is to determine a range of grid cells covering the query data. We define the range as *range of cells (RC)* which means the *ID* range of cells covering query data when the data is overlaid with grid cells. Thus, $RC =$

Query Data: reader = $rid_a - rid_b$, EPC_Pattern = $\langle m_1, [p_1-p_2], [s_1-s_2] \rangle$

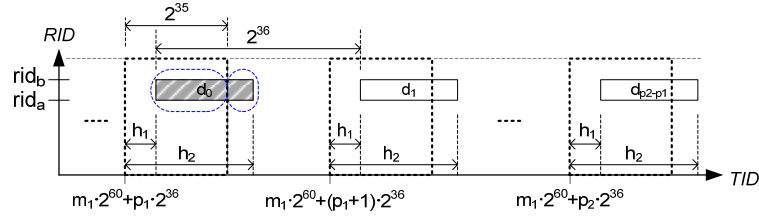
$$= \bigcup_{i=0}^{p_2-p_1} d_i = \bigcup_{i=0}^{p_2-p_1} \{(rid_a, m_1 \times 2^{60} + (p_1 + i) \times 2^{36} + s_1), (rid_b, m_1 \times 2^{60} + (p_1 + i) \times 2^{36} + s_2)\}$$



(a) Example: repeated pattern of cell length 2^{36} .



(b) Example: repeated pattern of cell length 2^{37} .



(c) Example: splitting query data.

Fig. 3. Repeated patterns in different cases.

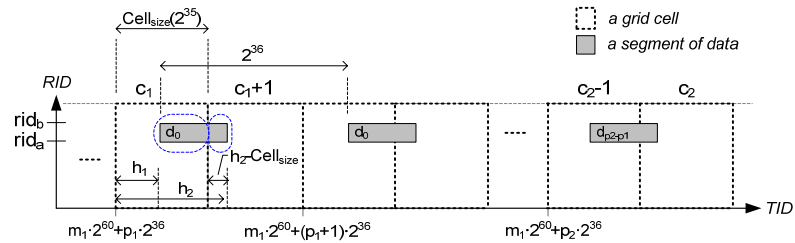
$\{\overline{(\min_{cid}, \max_{cid})} \subset R\}$. Let d_0, d_1, \dots , and $d_{p_2-p_1}$ be the segments of it. Then, the minimum value of range is the ID of the cell that contains the segment d_0 , and the maximum value is the ID of the cell that contains the segment $d_{p_2-p_1}$. These values are calculated by a hash function to find a cell ID using $(rid_a, d_0.min_{tid})$ and $(rid_b, d_{p_2-p_1}.max_{tid})$ as parameters.

The final step of the transformation is to create new *aggregated data (AD)* which is the representative of the segments of query data by using *RP* and *RC*. Initially, we define a transformed space composed of the RID axis, TID_{cell} axis and CID axis to get the aggregated data. RID and TID_{cell} mean the space of a grid cell and CID means the dimension of ID s used to distinguish cells of the virtual grid. The aggregated data in this transformed space is created by aggregating the *RP* in space (RID, TID_{cell}) and the *RC* on CID . We assume that the query data and the size of the grid cell are the same as described in Fig. 3 (a). We also assume that the *RP* is $\{(rid_a, h_1), (rid_b, h_2)\}$ and the *RC* is (c_1, c_2) . If we aggregate these in the transformed space, then the *RP* in space (RID, TID_{cell}) occupies from c_1 to c_2 on the CID axis. This region is denoted as

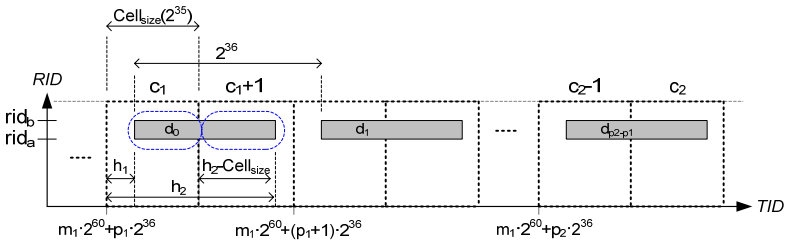
$$\bigcup_{i=c_1}^{c_2} \{(rid_a, h_1, i), (rid_b, h_2, i)\}. \tag{1}$$

We can create a new object covering the entire occupied space because all the elements of the equation are deployed on the *CID* axis in sequence. This object is the aggregated data of the query data and it is $\{(rid_a, h_1, c_1), (rid_b, h_2, c_2)\}$ by the corner representation.

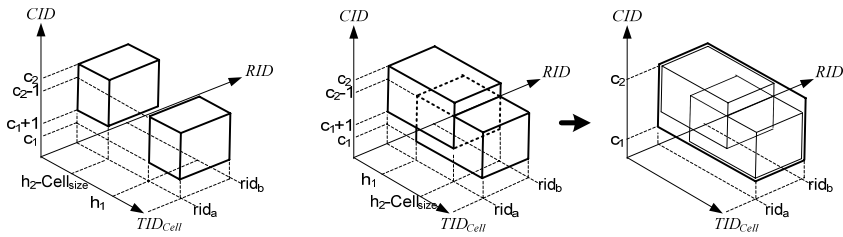
As shown in Eq. (1), transformation extends the repeated pattern on the *CID* axis. Thus, the number of repeated patterns generated from an input data determines the number of data in the index. Recall that a number of split patterns can be generated by overlapping segments of data with grid cells. Furthermore, the smaller the cell sizes the more the number of patterns. This phenomenon causes increase in the number of transformed data and growing of more and more overlapping regions among the data. Fig. 4 shows an approach to remove this problem occurred in the overlap case. No overlapped region is among the transformed data if the cell size is larger than the size of data segment as shown in the Fig. 4 (a). On the other hand, there are much overlapped regions among the transformed data if the cell size is smaller than the size of data segment as shown in the



(a) $h_2 - h_1 \leq \text{Cell}_{\text{size}}$ along the TID axis in the overlap case.



(b) $h_2 - h_1 > \text{Cell}_{\text{size}}$ along the TID axis in the overlap case.



(c) Transformed data of Fig. (a). (d) Transformed data of Fig. (b). (e) Transformed data using minimum bounding box of Fig. (b).

Fig. 4. Transformation processing for the overlap case.

Fig. 4 (b). For this reason, data is transformed separately depending on the former or the latter cases described in Figs. 4 (a) and (b) respectively. In the former case, each split pattern is transformed as shown in the Fig. 4 (c) that combines splitting policy and data aggregation mentioned previously. In the latter case, all the split patterns are transformed into a minimum bounding box (*MBB*) to prevent the problem occurred in the overlap case as shown in the Fig. 4 (e). Using this scheme in the overlap case between grid cells and segments of data, we can get two merits: at most two transformed data are only generated and no overlapping region is created from an input data.

Unifying the transformation steps, we can achieve the transform formula that allows the transformation processes to be treated more quickly. We assume that query data is composed of a range (rid_a, rid_b) on the *RID* axis and an *EPC* pattern $\langle m_1, [p_1 - p_2], [s_1 - s_2] \rangle$ on the *TID* axis. Let $d_0, d_1, \dots, d_{p_2-p_1}$ be the segments of it. Let $Cell_{size}$ be the length of the grid cell on the *TID* axis, c_1 be the *ID* of the cell containing $(rid_a, d_0.min_{tid})$, and let c_2 be the *ID* of the cell containing $(rid_b, d_{p_2-p_1}.max_{tid})$. Let h_1 and h_2 be the shortest and longest distance on the *TID* axis between a segment d_i and the bottom-left point of a cell containing the bottom-left point of d_i . The transform formulas are summarized in Table 2. Considering all the above mentioned cases, the formulas are categorized by three factors: (1) size of grid cell (2) geometric relation between the grid and data segments (3) cell size and segment size of data when the geometric relation is overlap. These factors can be found from following predicate logics (*PLs*):

- PL_1 : **shorter** when the cell size(2^K) of grid is equal to or shorter than the repeated gap of segments and **longer** in the other cases.
- PL_2 : **contains** when a grid cell contains each segment completely and **overlapped** in the other cases.
- PL_3 : **shorter** when the cell size is equal to or shorter than the size of data segment along the *TID* axis and **longer** in the other cases.

Table 2. Transform formulas for query data with format ReaderID = $rid_a \sim rid_b$ and EPC Pattern = $m_1, [p_1 - p_2], [s_1 - s_2]$ according to the predicates PL_1, PL_2 and PL_3 .

Case	PL_1	PL_2	PL_3	Possibility	Applied Formula
1	shorter	contains	shorter	impossible	–
2	shorter	contains	longer	possible	Formula ₁
3	shorter	overlapped	shorter	possible	Formula ₂
3	shorter	overlapped	longer	possible	Formula ₃ , Formula ₄
5	longer	contains	shorter	impossible	–
6	longer	contains	longer	possible	Formula ₅
7	longer	overlapped	shorter	impossible	–
8	longer	overlapped	longer	impossible	–
Formula List					
Formula ₁	$\{(rid_a, s_1, c_1), (rid_b, s_2, c_2)\}$				
Formula ₂	$\{(rid_a, 0, c_1), (rid_b, 2^K, c_2)\}$				
Formula ₃	$\{(rid_a, 0, c_1 + 1), (rid_b, h_2 - 2^K, c_2)\}$				
Formula ₄	$\{(rid_a, h_1, c_1), (rid_b, 2^K, c_2 - 1)\}$				
Formula ₅	$\{(rid_a, ((m_1 \cdot 2^{60} + p_1 \cdot 2^{36} + s_1) \text{MOD } 2^K) \text{MOD } 2^{36}, c_1), (rid_b, (((m_1 \cdot 2^{60} + p_1 \cdot 2^{36} + s_1) \text{MOD } 2^K) \text{MOD } 2^{36}) + ((2^{K-36} - 1) \cdot 2^{36}) + (s_2 - s_1), c_2)\}$				

From the three factors, we get eight different cases of syntactic combination. The detailed explanation of formulas about all cases is described as follows:

Case 1: This is an impossible case. It is always impossible that the two predicates $PL_2 = \text{'Contains'}$ and $PL_3 = \text{'Shorter'}$ are satisfied together.

Case 2: This case belongs to the first class of repeated pattern shown in the Fig. 3 (a). The intermediate equation to extract the repeated pattern is $\{(rid_a, s_1), (rid_b, s_2)\}$ and the range of cell is (c_1, c_2) . Therefore, the transform formula is $\{(rid_a, s_1, c_1), (rid_b, s_2, c_2)\}$. This formula holds in all cases because the maximum number of segments in a cell is one when $PL_1 = \text{'Shorter'}$ and $PL_2 = \text{'Contains'}$ are satisfied together.

Case 3: This case belongs to the form of repeated pattern introduced in the Fig. 3 (c) and is the same as the case shown in the Fig. 4 (b). Thus, the transform formula is $\{(rid_a, 0, c_1), (rid_b, 2^K, c_2)\}$ because a minimum bounding box (*MBB*) from the input data becomes the transformed data as shown in the Fig. 4 (e).

Case 4: This case belongs to the form of repeated pattern shown in the Fig. 3 (c) and is the same as the case shown in the Fig. 4 (a). Therefore, two split partitions generated due to overlapping of cell and input data. For each portion, the transform formula should be applied. As shown in the Fig. 4 (c), the first transform formula is $\{(rid_a, 0, c_1 + 1), (rid_b, h_2 - 2^K, c_2)\}$ and the second is $\{(rid_a, h_1, c_1), (rid_b, 2^K, c_2 - 1)\}$.

Case 5: This is also an impossible case. The reason is the same as case 1 mentioned above.

Case 6: This case belongs to the second class of repeated pattern shown in the Fig. 3 (b). If cell size grows more and more, the number of segments contained within a cell increases. Fig. 5 shows the general case where the predicates $PL_1 = \text{'Longer'}$ and $PL_2 = \text{'Contains'}$ and $PL_3 = \text{'Longer'}$ are true. Three segments are contained within the first cell but four segments are contained within the next and the latter cells. Thus, a *MBB* containing four segments becomes a repeated pattern. Thus, we should determine the left-most and right-most position of the *MBB* on the *TID* axis. First, the relative position of first segment d_0 within a cell is calculated by modulus of $m_1 \cdot 2^{60} + p_1 \cdot 2^{36} + s_1$ by cell size (2^K) and the left-most position of the *MBB* is calculated by modulus of the relative position of d_0 by the repeated gap (2^{36}). Recall that the number of segments within a cell is 2^{K-36} . We denote L is the distance from the left-most position of the *MBB* to the initial position of the last segment within the cell containing d_0 . This L could be found by multiplication of repeated gap and segment numbers occurred in a cell reduced by one. Thus, we get the value L is $(2^{K-36} - 1) \cdot 2^{36}$. The horizontal distance of the *MBB* is calculated by adding the L with the width of a segment ($s_2 - s_1$). Therefore, the right-most position of the *MBB* is the summation of the left-most position of the *MBB* and the horizontal distance of the *MBB* and the equation could be derived as $((m_1 \cdot 2^{60} + p_1 \cdot 2^{36} + s_1) \text{ MOD } 2^K) \text{ MOD } 2^{36} + ((2^{K-36} - 1) \cdot 2^{36}) + (s_2 - s_1)$. Finally, we get the transform formula is $\{(rid_a, ((m_1 \cdot 2^{60} + p_1 \cdot 2^{36} + s_1) \text{ MOD } 2^K) \text{ MOD } 2^{36}, c_1), (rid_b, (((m_1 \cdot 2^{60} + p_1 \cdot 2^{36} + s_1) \text{ MOD } 2^K) \text{ MOD } 2^{36}) + ((2^{K-36} - 1) \cdot 2^{36}) + (s_2 - s_1), c_2)\}$.

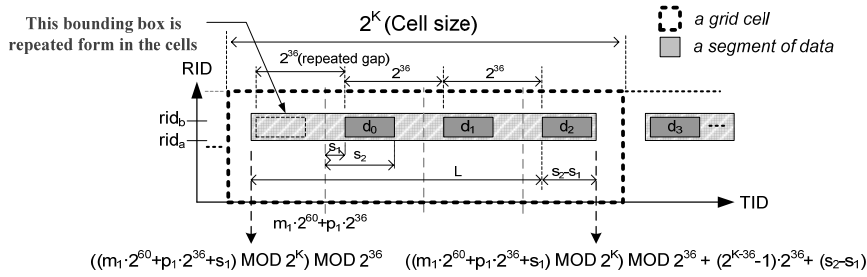


Fig. 5. A general example of the case 6 in the Table 2.

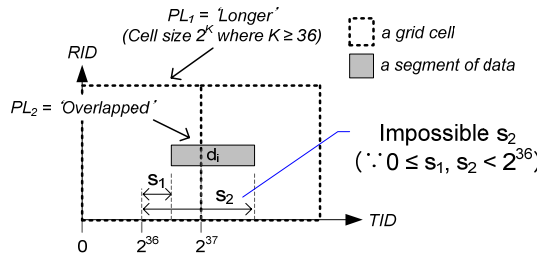


Fig. 6. An example of impossible case.

Cases 7 and 8: These are impossible cases. It is always impossible that the two predicates $PL_1 = \text{'Longer'}$ and $PL_2 = \text{'Overlapped'}$ are satisfied together. Serial field of any EPC pattern can have a value $[0, 2^{36} - 1]$ or a range formatted as $[s_1 - s_2]$ where $0 \leq s_1 < s_2 < 2^{36}$ according to the EPC pattern standard format. For example, Fig. 6 shows the case 7 or 8 where $PL_1 = \text{'Longer'}$ and $PL_2 = \text{'Overlapped'}$. To satisfy the existence of the segment d_i in the following figure, the s_2 of input data must have a value more than 2^{36} which is impossible.

In addition to above data transformation, it is also necessary to convert the stabbing query (rid, tid) into the transformed space. Note that the query is a point in the original space and a point is contained within a cell only. For this reason, the shortest distance (h_1) is equal to the longest distance (h_2) and c_1 is equal to c_2 . Thus, the transform formula for stabbing query becomes (rid_a, h_1, c_1) regardless of the predicate logics.

Without transformation, query index can be constructed in two-dimensional space (RID, TID) or in four-dimensional space $(RID, manager, product, serial)$. However, the former requires long insertion time and huge storage space to store all segments of complex query data whereas the latter causes deterioration of search performance due to data representation in higher dimensions. The advantage of our transformation is that it can outweigh the drawbacks of query index because it represents complex query data as simple data in three-dimensional space.

4. EXPERIMENTAL AND PERFORMANCE EVALUATION

In this section, we present experimental results of the proposed method. To apply the transformation, we extend insert and search algorithms of the R -tree that is widely

used for the multi-dimensional space, and the extended R -tree is denoted to R -tree (AT). We compare the performance of the R -tree (AT) with the original R -tree [19], CQI [4] and VCR [8] on various sets of data. The R -tree (AT) is a query index for the aggregate transformed space (RID, TID_{cell}, CID). The original R -tree is implemented as query index for four-dimensional space (RID , manager, product, serial), and CQI and VCR indexes are the existing query indexes in two-dimensional space (RID, TID). We applied approximations to the data of the CQI and VCR indexes because it is impossible to store all the segments of query data. All indexes are kept in the main memory to support real-time processing. The time we measured was the wall clock time. Our performance measurements were made on a standard personal computer with an Intel Pentium IV 2.6 GHz processor, 1GB of main memory and the Microsoft Windows Server 2003 operating system.

There are no well-known and widely accepted $RFID$ datasets for experimental purposes. Therefore, we carried out experiments with datasets, uniform and skewed distributions, from the *Research Center for Logistics Information Technology (RCLIT)*[‡]. Datasets of uniform distributions are always ruled by a random distribution achieving unbiased data in the space and the datasets consist of 5K, 10K, 50K and 100K entries. Skewed distributions are positively skewed in detail and these also consist of 5K, 10K, 50K and 100K entries. The datasets are constructed as parameters: the center point is 5% distance from the origin to both domain axes (RID, TID) and the maximum deviation is 0.05, thus all the data appear always within the 5% over distance from the center point. For both distributions, the EPC patterns of input data are randomly chosen from Table 1. To measure the search performance, here we used 10K random stabbing queries.

Firstly, we attempted to identify an effective cell size for the transformation considering dataset of uniform distributions consists of 5000, 10,000, 50,000 and 100,000 query data. The effective cell size is 2^{36} because the RC includes empty cells if cell size is smaller than 2^{36} and the RP contains empty regions if cell size is larger than 2^{36} . It is not absolute optimal value but efficient value calculated experimentally. In this paper, we use the cell size of 2^{36} as the default.

We also carried out experiments of insertion performance. Fig. 7 shows the insertion performance for various distributions of the dataset. Overall, the grid-based CQI index outperforms the others because the insertion process of the index is very simple and intuitive. On the other hand, the VCR index shows lower performance than others because it generates many partitions of data and stores them individually in the insertion process. The insertion performance of the R -tree is similar to the R -tree (AT). However, the CQI and VCR indexes require refinement step in query processing because they store approximate data.

Finally, we measured the performance for the stabbing query of the indexes where the number of stabbing queries is 10,000. Fig. 8 shows the search performance for uniform and skewed distributions of the dataset. The CQI and VCR indexes take a long time to processing queries because they must identify every data sequentially in the cell or VC s containing the point of query. Thus, the search performance of these indexes deteriorates much more quickly as the number of data grows larger. Overall, the R -tree (AT) performs better than the R -tree. The reason is that the R -tree (AT) searches data in three-dimensional space but the R -tree searches data in four-dimensional space.

[‡] The *RCLIT* is Korea national project for developing the next generation of logistics information technology. It focuses on logistics research and practice with IT-based technology such as ubiquitous computing and $RFID$ system.

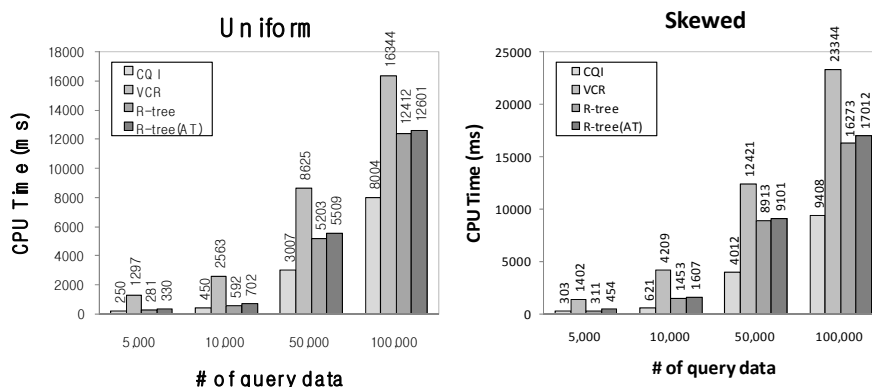


Fig. 7. Insertion costs.

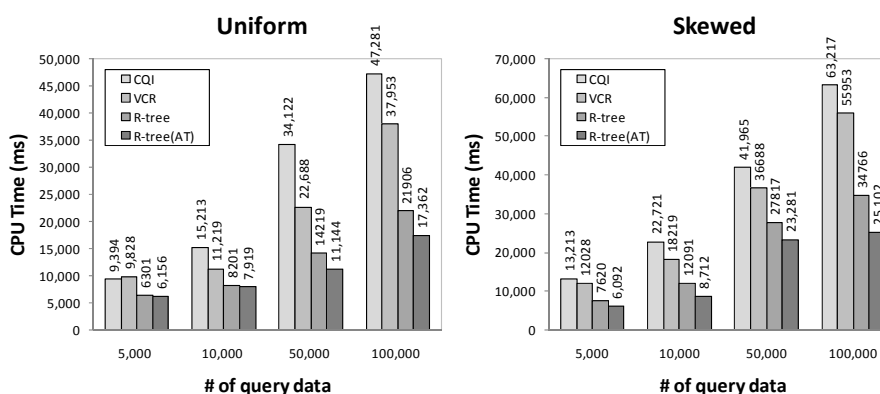


Fig. 8. Search costs.

5. CONCLUSIONS

Query indexes in previous studies have treated a simple query as data, such as a region continuous query or an interval continuous query. However, data in the *RFID* query index is represented as many segments because it is a continuous query from the *ECSpec*. With any of the existing methods as the *RFID* query index, it is very time consuming to build an index on these data, because it is necessary to insert a great many segments into the index for storing a continuous query. The index size also makes it inefficient to process stabbing queries. In this paper, we proposed a technique for indexing complex continuous queries. We first established the properties of congruence and repetition between the segments of the continuous query. Based on these properties, we suggested a transform technique, called *aggregate transformation*, which finds a group from the huge segments and changes the group of segments into aggregated data. The aggregated data is representative of the segments. That is, it allows the query index to store one object instead of inserting all the segments. The experiment results show the performance improvement by the proposed technique, compared with existing query indexes. This study focused on the *GID-96* class 1 tag type only. It is necessary to support various tag types

such as SGTIN, SGLN *etc.* Recently, tags following the GEN2 standard have been introduced and they have memory to store security information, product information, user information *etc.* Our future study will be related to support a wide variety of tags following GEN2 standard.

ACKNOWLEDGMENT

This work was supported by the grant of the Korean Ministry of Education, Science and Technology. (The Regional Core Research Program/Institute of Logistics Information Technology)

REFERENCES

1. EPCglobal, *The Application Level Event (ALE) Specification Version 1.0*, EPCglobal Standard Specification, 2005.
2. EPCglobal, *EPCTM Tag Data Standards Version 1.3*, EPCglobal Standard Specification, 2005.
3. V. Gaede and O. Günter, "Multidimensional access methods," *ACM Computing Surveys*, Vol. 30, 1998, pp. 170-231.
4. D. V. Kalashnikov, S. Prabhakar, W. G. Aref, and S. E. Hambrusch, "Efficient evaluation of continuous range queries on moving objects," in *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, Vol. 2453, 2002, pp. 731-740.
5. S. Madden, M. Shah, J. M. Hellerstein, and V. Raman, "Continuously adaptive continuous queries over streams," in *Proceedings of the ACM SIGMOD*, 2002, pp. 49-60.
6. J. T. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexes," in *Proceedings of the ACM SIGMOD*, 1981, pp. 10-18.
7. Y. Theodoridis and M. A. Nascimento, "Generating spatiotemporal datasets on the WWW," *ACM SIGMOD Record*, Vol. 29, 2000, pp. 39-43.
8. K. L. Wu, S. K. Chen, and P. S. Yu, "Processing continual range queries over moving objects using VCR-based query indexes," in *Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems*, 2004, pp. 226-235.
9. Y. Bai, F. Wang, and P. Liu, "Efficiently filtering RFID data streams," in *Proceedings of the 1st International VLDB Workshop on Clean Databases*, 2006, pp. 50-57.
10. F. Wang and P. Liu, "Temporal management of RFID data," in *Proceedings of the 31st Very Large Data Bases Conference*, 2005, pp. 1128-1139.
11. J. Chen, *et al.*, "NiagaraCQ: A scalable continuous query system for internet databases," in *Proceedings of the ACM SIGMOD*, 2000, pp. 379-390.
12. S. Chandrasekaran, *et al.*, "TelegraphCQ: Continuous dataflow processing for an uncertain world," in *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research*, 2003, pp. 269-280.
13. E. N. Hanson, *et al.*, "A predicate matching algorithm for database rule systems," *ACM SIGMOD Record*, Vol. 19, 1990, pp. 271-280.
14. S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma, "Managing RFID data," in *Proceedings of the 30th International Conference on Very Large*

- Data Bases*, 2004, pp. 1189-1195.
15. S. Sarma, "Integrating RFID," *ACM Queue*, Vol. 2, 2004, pp. 50-57.
 16. L. Golab and M. T. Oszu, "Issues in data stream management," *ACM SIGMOD Record*, Vol. 32, 2003, pp. 5-14.
 17. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.
 18. C. Böhm, S. Berchtold, and D. Keim, "Searching in high-dimensional spaces-index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, Vol. 33, 2001, pp. 322-373.
 19. A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the ACM SIGMOD*, 1984, pp. 47-57.
 20. J. K. Park, B. H. Hong, and C. H. Ban, "A continuous query index for processing queries on rfid data stream," in *Proceedings of the 13th International Conference on Embedded and Real-Time Computing System and Applications*, 2007, pp. 138-145.

Jaekwan Park received the M.S. and Ph.D. degrees in Computer Engineering from Pusan National University, Busan, Korea, in 2001 and 2008. He is currently a senior research engineer in the LGE Advanced Research Institute, Seoul, Korea. He has developed RFID middleware with Research Center for Logistics Information Technology. His research interests include RFID system, RFID middleware and sensor network system.

Bonghee Hong received the M.S. and Ph.D. degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1984 and 1988. In 1987, he joined the faculty of Computer Engineering of the Pusan National University (PNU). He is working as a Professor of Database in the Department of Computer Engineering at the PNU. His research interests include theory of database systems, moving object databases, spatial databases, RFID system and RFID middleware.

Chaehoon Ban received the M.S. and Ph.D. degrees in Computer Engineering from Pusan National University, Busan, Korea, in 1999 and 2006. He is currently a professor at Department of Internet Business, Kosin University. His research interests include spatial index, moving object index and RFID system.