Improvement of Filtering Algorithm for RFID Middleware Using KDB-tree Query Index

Xiaobo Zhang
Faculty of Automation, Guangdong University of Technology, Guangzhou 510006, China
Email: zxb_leng@163.com

Lianglun Cheng
Faculty of Automation, Guangdong University of Technology, Guangzhou 510006, China
Email: lcheng@gdut.edu.cn

Quanmin Zhu
Intelligent Autonomous Systems Lab, University of the West of England, Bristol,
Coldharbour Lane, Bristol BS16 1QY, UK
Email:QuanZhu@uwe.ac.uk

***Abstract*—RFID middleware collects and filters RFID streaming data gathered continuously by numerous readers to process requests from applications. These requests are called continuous queries. The problem when using any of the existing query indexes on these continuous queries is that it takes a long time to build the index because it is necessary to insert a large number of segments into the index. KDB-tree is an index which can dispose multidimensional data. It is also a dynamic balance tree that has a good query performance and high spatial usage. This paper propose an aggregate transformation algorithm for querydata filtering, and applies KDB-tree into RFID event filtering to improve the performance of query. Comparing to other indexes, the result of simulation shows that KDB-tree index outperforms others in synthesized consideration of storage cost, insertion time cost and query time cost. In particular the query time cost of KDB-tree is distinctly lower than others because it provides single path traverse in the query process.**

***Index Terms*—RFID middleware; filtering algorithm; KDB-tree; querydata; pointquery**

I. INTRODUCTION

RFID middleware collects and filters the data transferred by readers. This work will bring high load for RFID system with two reason, one is the reader's continual action will produce continual data, the other is multiple ECSpec prescribes multiple query processes for RFID events and these query processes could make RFID middleware continually filter tag data according to multiple conditions. For these reasons, RFID middleware must have a good algorithm in order to filter events from tag data quickly and accurately.

To present, the researches about event filtering of RFID middleware mostly focused on the theory of memory database[1-3]. The method is to search better index to improve the query process. A better way for improvement of the query process is to construct various multidimensional indexes[4-5]. There are many indexes could be applied into RFID event filtering such as Hash,CQI,VCR,R-tree etc. CQI(*Cell-based Query Index*) is a kind of index on the basis of memory[6]. In such index, query scope is divided into various cells. Every cell has a query table which intersects or joins query conditions, the cell uses this query table to filter results. VCR(*Virtual Construct Rectangle*) index is also a kind of index on the basis of memory[7]. In such index, query scope is divided into multiple segments. These segments are encapsulated by VCs(*virtual constructs*) with inserted ID. We can search these VCs for results. CQI and VCR both are two-dimensional space composed of RID(*Reader Identification Domain)* and TID(*Tag Identification Domain)* ,and both regard constant query as a whole domain, while ECSpec is not a domain but segment. If such indexes were applied into RFID query, there must be enough memory for distribution and the times of insertion will be high. An improvement method is to use multidimensional indexes such as R-tree [8-10]. While in such index, there are four-dimensional data should be stored: RID, manager, product and serial number [11]. For this reason, the performance of this index will decline for high dimensions. As we know, the query performance of the index deteriorates exponentially with higher dimensions [12]. Thus, the traditional indexes can decrease many insertions of querydata, but they are not efficient for query.

Due to the disadvantages of these multidimensional indexes, this paper provides a new index like KDB-tree for RFID middleware [13]. KDB-tree is also a kind of

This work was supported in part by the Guangdong Natural Science Foundation(#8351009001000002 and #07117421), the Major Program of Guangdong Natural Science Foundation(#2009A080207008), the Joint Funds of the National Natural Science Foundation of China and Guangdong Natural Science Foundation(#U0935002) , the Youth Foundation of Guangdong University of Technology(#405095245).
Corresponding author.: zxb_leng@163.com (Xiaobo Zhang)

doi:10.4304/jsw.6.12.2521-2527

multidimensional index composed of RID, SID and TID, but it can decrease the number of insertion and the space cost, and also keep the tree in balance. Besides these, in KDB-tree index, a pointquery has only one path from root to leaf.

II. AGGREGATION AND CONVERSION OF QUERYDATA

Definition 1*: RFID middleware collects and filters a tagdata from reader for once. This process is called a pointquery, the expression can be described as PointQuery (RIDi, TIDj).*

Definition 2*: querydata is a continual query based on ECSpec and is a filtering condition for reader and tag. RID is the scope of readers and TID is the filtering condition for tags.*

Definition 3*: querydata with only one segment is called simple querydata and querydata with two or more segments is called complicated querydata.*

Querydata includes one or more segments which are composed of RID and TID in two-dimensional space. A segment is the smallest unit in complicated querydata. if querydata is set to d (d={d1,…,dn}, $1<=n<=2^{24}$), then the segment of querydata could be set to di={(minrid,mintid),(maxrid,maxtid)}, di∈d.

In two-dimensional space of (RID, TID), the querydata is composed of discrete segments. If user wants to search “the Nokia cellphones made in this year in warehouse A”, the system will send an ECSpec request including CQ1: “readerID=2,EPC-Pattern=<10.[1-3].[3001-4000]>”[14]. It shows that the readerID in warehouse A is 2, CQ1 arrives RFID middleware and is inserted into query index, then the querydata has three discrete segments, the first is {(1, 10·260+ 1·236 + 3001), (1, 10·260+ 1·236+ 4000)}, the second is {(1, 10·260+ 2·236+ 3001), (1, 10·260+ 2·236+ 4000)}, and the last is {(1, 10·260+ 3·236+ 3001), (1, 10·260+ 3·236+ 4000)}.

From above we can see that if a product ID P with EPC mode is a scope, the querydata will include multiple segments. The size of the segment is determined by the scope of serial number. Querydata is a complicated object which includes 2^{24} segments at most. When a continual ECSpec query is executing, we must insert segments into index. Too many times of insertion will make index much larger and the performance of pointquery will decline.

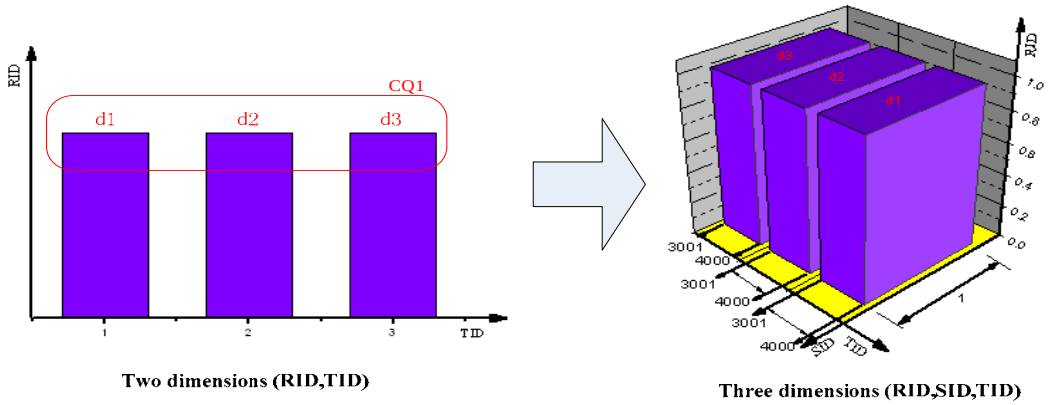
To reduce the insertion quantity, the number of segments must be declined. This paper provides an aggregation and conversion method that transform two dimensional data (RID,TID) to three dimensional data (RID,SID,TID). ECSpec provides 27 different modes while only 11 modes have actual significance. The querydata is determined by EPC mode which includes Manager, Serial Number and Product. In EPC mode, every part is a constant number, or [low-high] ,or *.

Definition 4*: there are three dimensionalities:(s,t are two dimensionalities in two-dimensional space.)*

S: $m*2^{24}+s$;
R:RID;
T:t

Then any querydata in this three-dimensional space can be expressed as one segment: d={(minRID,minSID,minTID),(maxRID,maxSID,maxTID)}.

Every querydata only need to be transformed into a three-dimensional query segment by RFID middleware (see in Fig.1.). In this way, the quantity of insertion could be declined.



(CQ1: readerID=2,EPC-Pattern=<10.[1-3].[3001-4000]>)
Fig.1. aggregation and conversion of querydata

III. FILTERING ALGORITHMS OF KDB-TREE

A. Data Structure of KDB-tree Index

Like B-tree, KDB-tree consists of a collection of pages and a variable root ID that gives the page ID of the root page. There are two types of pages in a KDB-tree:

1)Region pages: region pages contain a collection of pairs(*region, page ID*).

2)Point pages: point pages contain a collection of pairs(*point, location*), in which *location* gives the location of a database record. The pair (*point, location*) is an index record.

KDB-tree has following properties:

1)Considering each page as a node and each page ID in a region page as a node pointer, the resulting graph structure is a tree with root *root ID*. Furthermore, no region page contains a null pointer, and no region page is

empty.

2)The path length, from root page to leaf page is the same for all leaf pages. That is, the KDB-tree is a balance tree.

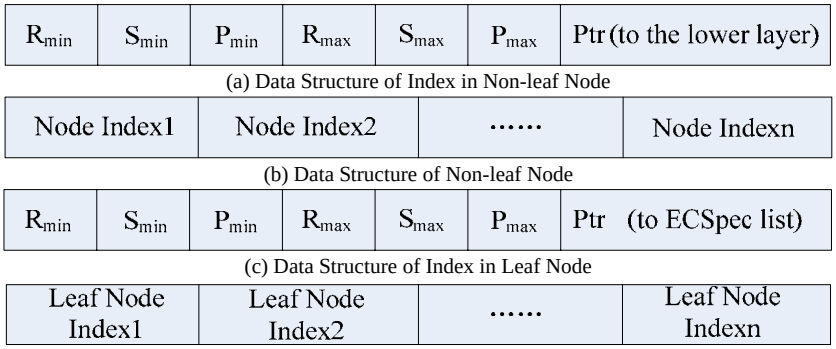
3)In every region page, the regions in the page are disjoint, and their union is a region.

4)If the root page is a region page, the union of its regions is *domain0*×*domain1*×…×*domainK-1*.

5)If (*region,child ID*) occurs in a region page, and the child page referred to by *child ID* is a region page, then the union of the regions in the child page is *region*.

6)Referring to (5), if the child page is a point page, then all the points in the page must be in *region*.

In RFID middleware, KDB-tree index is a three-dimensional structure with the three-dimensional data aggregated and conversed from two-dimensional data, including non-leaf node structure and leaf node structure. The items of non-leaf node store three dimensional information {(Rmin，Smin，Pmin)，(Rmax，Smax,Pmax)} and a pointer to the lower layer, while the items of leaf node store the query scope of ECSpec {(Rmin，Smin，Pmin)，(Rmax，Smax,Pmax)} and a pointer to the ECSpec list. As is shown in Fig.2.



(a) Data Structure of Index in Non-leaf Node

(b) Data Structure of Non-leaf Node

(c) Data Structure of Index in Leaf Node

(d) Data Structure of Leaf Node

Fig.2. data structure of KDB-tree index

B. Pointquery Algorithm of KDB-tree Index

KDB-tree is a kind of multidimensional index, and has a higher query performance, especially in pointquery. KDB-tree provides a kind of traverse with single path. Every pointquery has one path from root to leaf, and can always keep the tree balance in dynamic insertion.

While RFID middleware collects a tagdata from RFID reader and transforms it into three-dimensional data (RID,SID,TID), then searches the suitable querydata in KDB-tree index. This process is called a pointquery. The process can be described with function PointQuery(root,d) in which d is a querydata. Following is the pointquery algorithm:

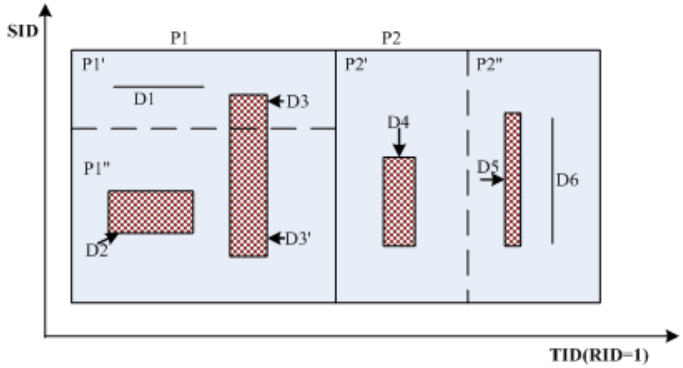
Algorithm 1: PointQuery(root,d)

```
PointQuery(root,d)
{
  If root is NULL return FALSE;
    for each entry root.e {
      If Contain(root.e,p) is TRUE  {
        If root is a leaf     Return TRUE;
        If root is not a leaf   PointQuery(e,d);
      }
    }
}
```

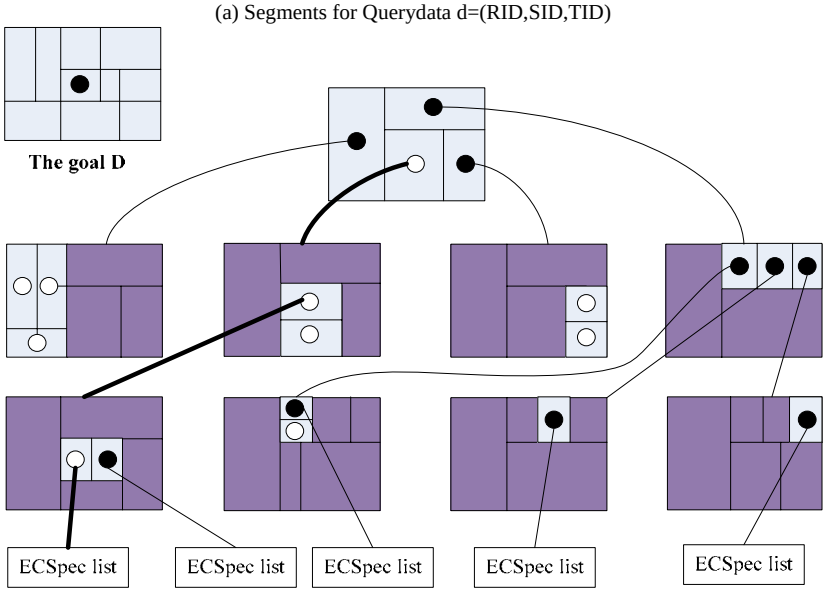
Pointquery starts from root. If the visited node is NULL, it shows that pointquery did not find the querydata segment with d, then it traverses every node in prior order, transfer *Contain(e,p)* for every segment *e*. If the result is TRUE, pointquery will go on judging whether it is a leaf node. If it is a leaf node, then it indicates that the pointquery has found the querydata segment with *d*, otherwise recursivly transfer the pointquery procedure.

Fig.3 is an example of pointquery. In Fig.3(a), the rectangle represents every querydata segment. Every querydata segment could be displayed in two-dimensional space due to RID=1. Then RFID middleware transfers the function *transform()* to converse (RID,TID) to d=(RID,SID,TID) before pointquery procedure.

In Fig.3(b), the rectangle in the left top expresses the position of the goal D in KDB-tree index and its space scope. Every other rectangle expresses an index item, the size and position of the rectangle is the space scope and the position in the KDB-tree index. Several index items compose to a node, like the colored parts in the rectangle. In leaf node, the rectangle with a dot expresses the scope of the querydata. The bold line is the path of the goal D in traversing KDB-tree. When in traversing, we should start from root node, find the index item with the scope of goal D and visit the next node by the pointer in index item. Repeat this process till we find the index item with the scope of goal D in leaf node. Then we can visit ECSpec list by the pointer in this index item and the pointquery ends.



(a) Segments for Querydata d=(RID,SID,TID)



(b) Process of Pointquery

Fig.3. Example of Pointquery

C. Insertion Algorithm of KDB-tree Index

When RFID middleware accepts an ECSpec request, it converses the querydata into a node N, as is shown in the left top in Fig.4. The insertion procedure is composed of two steps, the transformation step and the insertion step. Before querydata is inserted into the query index, it should firstly be converted into aggregated data. This process is executed by the transform function. After conversion, the insertion process of the aggregated data is the same as the original KDB-tree algorithm. The insertion algorithm of KDB-tree index could be described as following:

Algorithm 2:Insertion Algorithm

①Firstly traverse the index from the root node with single path till a leaf node LN.

②If LN has already had an index item, and its scope is the node N's scope, then a new ECSpec1 list should be added to the existing ECSpec list.

③Otherwise find an index item which scope includes the node N's scope and execute the split algorithm.

④Adjust the scope of the index item and add a new ECSpec1 list to its existing ECSpec list.

⑤Create a new node newLN which scope equals the node N's scope, and create a new ECSpec list newEL and insert to ECSpec1.

⑥make newLN point to newEL.

The split strategy is based on two aspects. The first is to determine an efficient split axis in (RID,SID,TID). The complexities of the axes could be calculated in the node. The complexity of an axis is the ratio of data to the space of the axis when the data in the node are projected onto the axis. The complexity of an axis is a value dividing the summation of lengths of data in the node by the entire length of the node on the axis. We choose the axis that has the minimum complexity value as the split axis because an uncomplicated axis minimizes the number of duplications of data.

The second is to determine an effective split position on the split axis. It considers dispersion and duplication. The degree of dispersion is the distribution ratio of data in the overflow node. The degree of duplication is the amount of data stored in both sides of the split. The equation that finds the split position by accommodating these two factors is as following.

$$SP_i = e \cdot D_1 + (1-e) \cdot \left(\frac{1}{2}\right)^{D_2} \quad (0 \le e \le 1) \tag{1}$$

SPi is the *ith* split position on the split axis, *D1* is the distributed ratio of data in the overflow node, *D2* is the number of data stored on both sides of the split, and *e* is a constant value to give more weight between the two factors. According to Equation 1, the SP value increases

as the degree of dispersion (*D1*) increases and the degree of duplication (*D2*) decreases. We choose the maximum *SPi* from the values calculated by Equation 1. Then, the data in the overflow node are distributed more uniformly and duplicated less by the split. Following is the split algorithm:

Algorithm 3: Split(N_{old}, N_{new}, obj)

axis = FindSplitAxis(N_{old}, obj);

sp = FindSplitPosition(N_{old}, obj, axis);

if N_{old} is leaf node

store obj into sp on axis in N_{new} and delete them in N_{old};

else{

store MBBs into sp on axis in N_{new} and delete them in N_{old};

add the entry pointing N_{new} to parent node of N_{old};

}

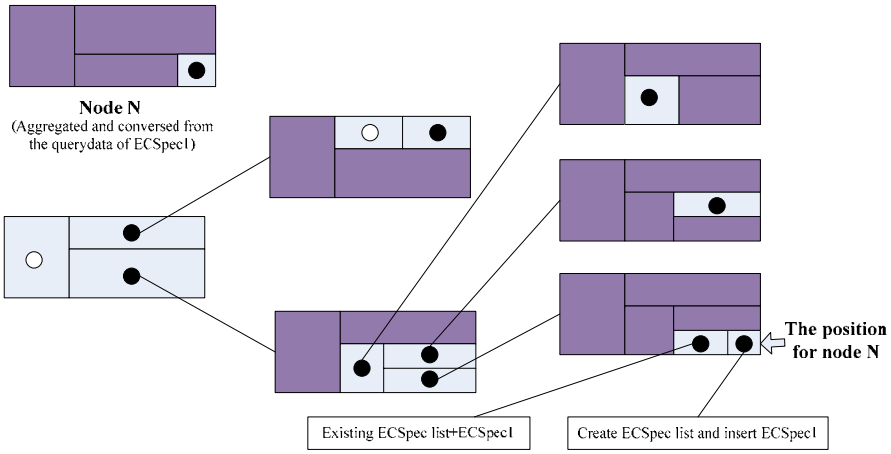


Fig.4. Insertion process of KDB-tree

IV. Simulation and Performance Comparison

In this simulation, we do a comparison with KDB-tree、R-tree、CQI and VCR index. KDB-tree index is based on three-dimensional space (RID,SID,TID), R-tree index is based on four-dimensional space (RID, Manager, Product, Serial number), while CQI and VCR are based on two-dimensional space (RID, TID). These indexes are stored in memory and use *Wall Clock Time* as measurement. The simulation of these indexes use the same ECSpec data and pointquery data. The comparison of these indexes is in three aspects: storage costs, insertion time costs and comprehensive performance.

The simulation platform of RFID event filtering consist of virtual reader, virtual client and RFID middleware. To reduce the effect of the virtual reader and virtual client on RFID event filtering, the structure of system is distributed mode. That is, virtual reader, virtual client and RFID middleware runs in different computers in LAN. The communication between virtual reader(virtual client) and console platform is implemented by web service.

The first comparison is the storage costs of KDB-tree、R-tree、CQI and VCR by using 5000, 10000, 50000, 100000 querydata to insert, as is shown in Fig.5. The result shows that CQI and VCR need the largest storage for the querydata must be separated into multiple segments to be inserted into index. R-tree needs the smallest storage. It transforms a querydata into four-dimensional data and inserts the data into index only once. KDB-tree needs more storage than R-tree, and transforms a querydata into three-dimensional data.

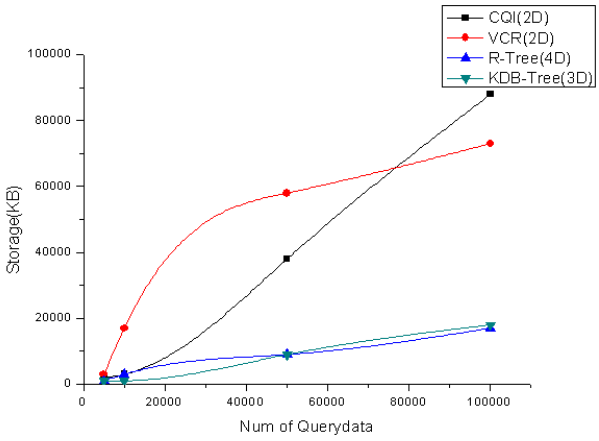


Fig.5 comparison of storage costs

The second comparison is the insertion time costs of KDB-tree、R-tree、CQI and VCR by using 5000, 10000, 50000, 100000 querydata to insert, as is shown in Fig.6. The insertion time for CQI is lowest, whereas the insertion time for VCR is highest for the querydata must be separated into multiple segments. KDB-tree and R-tree have the same time costs for insertion.

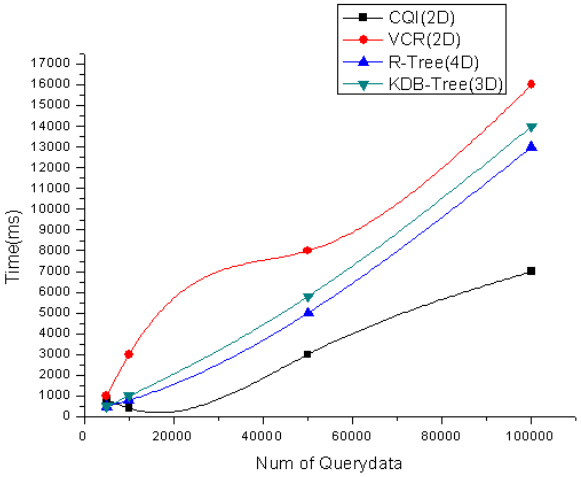


Fig.6 comparison of insertion time costs

The last comparison is the comprehensive performance, as is shown in Fig.7. The comprehensive performance of CQI and VCR is lowest for they need more time to insert than other indexes. KDB-tree's comprehensive performance is better than R-tree's for KDB-tree could traverse every node by single path while R-tree needs multiple paths to traverse. The result of the simulation shows that KDB-tree is the best query index in the index family.

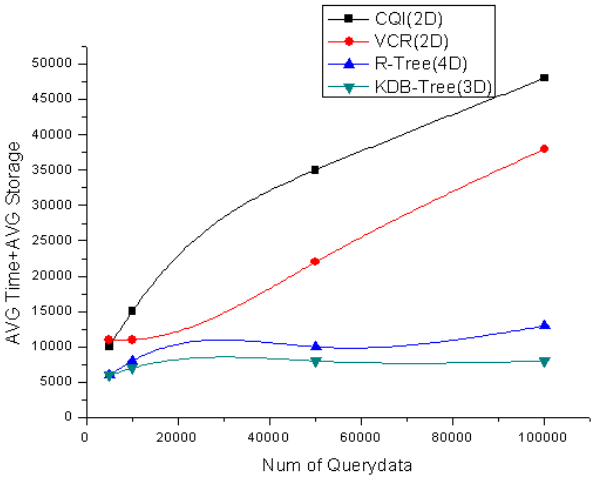


Fig.7 comparison of query performance

V. CONCLUSION

The query index approach is usually suitable for evaluating continuous queries over streaming data. Query indexes in previous studies have treated a simple query as the data, such as a region continuous query or an interval continuous query. However, the data in the RFID query index is represented as a great 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 queries.

The paper provides an aggregation and conversion method to querydata and a KDB-tree index for RFID middleware. There are several indexes for RFID event filtering, but they also have their shortages in some aspects. The result of simulation shows that KDB-tree has a comparatively good performance than other indexes. The algorithm based on KDB-tree index could be applied into actual RFID system to make RFID middleware filter large tagdata more quickly and accurately. So the performance of RFID system could be improved.

ACKNOWLEDGMENT

We would like to express our great appreciation to all the members of our laboratory for their technical insights and stimulating ideas, which greatly contributed to the success of our research.

REFERENCES

[1] YU Jian,LAI Sheng-li. Structure of main memory databases of radio frequency identification middleware. *Journal of Harbin Engineering University*.2008,Vol. 29,no. 6, pp.578-582

[2] Rong-Jhang Liao, Pei-Lun Suei, Yung-Feng Lu, Tei-Wei Kuo, Chun-Sho Lee. A Signature-based Grid Index Design for RFID Main-Memory Databases. *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008. EUC '08.* Vol. 1, pp.519-525

[3] Bin Cui, Beng Chin Coi, Jianwen Su, Tan K.-L. Indexing high-dimensional data for efficient in-memory similarity search. *IEEE Transactions on Knowledge and Data Engineering*. 2005,Vol. 17, Issue. 3, pp. 339-353

[4] Jaekwan Park, Bonghee Hong, Chaehoon Ban. Efficient Transformation Scheme for Indexing Continuous Queries on RFID Streaming Data. *Second International Conference on Systems and Networks Communications, 2007. ICSNC 2007*, pp.41-47

[5] Jaekwan Park, Bonghee Hong, Chaehoon Ban. A Continuous Query Index for Processing Queries on RFID Data Stream. 13th IEEE International *Conference on Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007*, pp. 138-145

[6] D.V Kalashnikov, S.Prabhakar W.G. Aref and S.E Hambrusch. Efficient evaluation of continuous range queries on moving objects. *Proc.13th International Conference on Database and Expert Systems Applications,2002*, pp.731-740

[7] K. L Wu, S. K Chen and P. S Yu. Processing Continual Range Queries over Moving Objects Using VCR-Based Query Indexes. *International Conference of Mobile and Ubiquitous Systems . 2004*, pp.226-235

[8] Zhenwen He, Chonglong Wu, Cheng Wang. Clustered Sorting R-Tree: An Index for Multi-Dimensional Spatial Objects. *Fourth International Conference on Natural Computation, 2008. ICNC '08.* Vol. 6, pp. 348-352

[9] Proietti G., Faloutsos C. Analysis of range queries and self-spatial join queries on real region datasets stored using an R-tree. *IEEE Transactions on Knowledge and Data Engineering*. 2000,Vol. 12 , Issue. 5, pp. 751-762

[10] Weihua Lin, Yonggang Wu, Xiaojun Tan, Yan Yu. An Improvement of Index Method and Structure Based on R-Tree. *2008 International Conference on Computer Science and Software Engineering* . Vol 4, pp. 607-610

[11] EPCglobal, EPC Tag Data Standards Version 1.3. EPCglobal Standard Specification[S], 2005.

[12] Lifang Yang, Rui Lv, Xianglin Huang, Yueping Liu.

Performance of R-Tree with Slim-Down and Reinsertion Algorithm. *International Conference on Signal Acquisition and Processing, 2010. ICSAP '10*, pp. 291-294

[13] Hung-Yi Lin, Po-Whei Huang. Perfect KDB-Tree: A Compact KDB-Tree Structure for Indexing Multidimensional Data. *Third International Conference on Information Technology and Applications, 2005. ICITA 2005.* Vol. 2, pp. 411-414

[14] EPCglobal, The Application Level Event (ALE) Specification Version 1.1[S], EPCglobal Standard Specification, 2008.

Xiaobo Zhang Lecturer and doctor candidate at the Faculty of Automation, Guangdong University of Technology. His research interests are embedded system, wireless sensor networks, dynamic model and intelligent control.

Lianglun Cheng Prof. and doctoral supervisor of the Faculty of Automation, Guangdong University of Technology. His research interests are network control and integration, information control, embedded system.

Quanmin Zhu Prof. in control systems at the Faculty of Computing, Engineering and Mathematical Sciences(CEMS), University of the West of England (UWE), Bristol, UK. His research interests are nonlinear system modeling, identification, and control.