LETTER

Indexing of Tagged Moving Objects over Localized Trajectory with Time Intervals in RFID Systems**

Jongwan KIM[†a)], ***Member***, **Dukshin OH**[†], ***and*** **Keecheon KIM**[††*], ***Nonmembers***

SUMMARY Since a radio frequency identification (RFID) transponder (tag) generates both location and time information when it enters and leaves a reader, the trajectory of a moving, tagged object can be traced. Due to the time intervals between entries to successive readers, during which tags are not tracked, accurate tracing of complete trajectories can be difficult. To overcome this problem, we propose a tag trajectory indexing scheme called TR-tree (R-tree-based tag trajectory index) that can trace tags by combining the local trajectories at each reader. In experiments, this scheme showed superior performance compared with other indices.

key words: *R-tree, trajectory, spatiotemporal index, RFID, tag index*

1. Introduction

The radio frequency identification (RFID) system consists of RFID readers, radio frequency (RF) tags, and computer systems. A tag uniquely identifies an object [1]. RFID tags are used for tracking inventory, container management at harbors, supply chain management, *etc*. One reason for attaching tags to objects is tracing [1], [2]. To trace and manage the movement of tagged objects requires a trajectory index that specifies both previous and current movements.

A tag contains spatiotemporal information that shows the location and time of entry at each reader. This information can be managed in a manner similar to that in existing spatiotemporal data-management methods for moving objects. However, despite the shared characteristics of RFID tags and other spatiotemporal data, there has not been sufficient research on RFID tag trajectories.

Tags differ from existing moving objects in two ways. First, a moving object normally reports its location to a server regularly using a location identification device such as a global positioning system (GPS) [7]. An RFID tag, however, does not possess a device for identifying its location. Second, a moving object has information about its location continuously, whereas a tag's location is known only when it is read by a reader. Despite these differences, it is possible to manage a tag's trajectories using only information from readers.

Manuscript received December 25, 2009.

[†]The authors are with the Dept. of Management Information Systems, Sahmyook University, Seoul, Korea.

[††]The author is with the Dept. of Computer Science & Engineering, Konkuk University, Seoul, Korea.

[*]Corresponding author.

[**]This work was supported by the National Research Foundation of Korea Grant funded by the Korean Government [NRF-2009-351-D00075].

a) E-mail: wany@korea.ac.kr

DOI: 10.1587/transinf.E93.D.2639

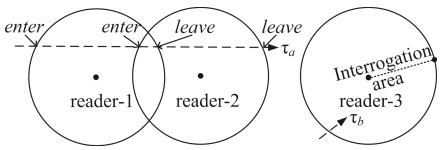


Fig. 1 Movement and interrogation of tags (τ_a, τ_b).

Due to the characteristics of RFID systems, a tag is localized by the reader's location, and the time when the tag enters the reader and leaves the reader can be checked. Since a tag's time trajectories are tracked separately by each reader, there are intervals between the time trajectories tracked by readers. This interruption affects the generation of tags' trajectories, making it difficult to trace their past trajectories. To overcome this problem, in this paper we construct complete trajectories for tags by connecting the local trajectories created by individual readers. We propose TR-tree, a tag index based on R-tree [3] that can trace tags' movements by composing minimum bounding boxes (MBBs) along the dimensions of location and time in the index. We implement TR-tree with a tag-link (TL) index that is a hash-based index for costless dynamic updates [4].

We assume that tags are passive tags that receive signals from readers and transmit data via backscattering [1]. As shown in Fig. 1, an *enter event* occurs when a tag, τ_a or τ_b, enters a reader's area, and a *leave event* occurs when it leaves that area. By using these two events, we can calculate how long the tag has stayed in the reader's area.

Our Contributions. We have developed a tag trajectory index without time interruptions to trace tags in RFID systems. Our contributions are as follows. First, we use R-tree-based trajectory index to manage RFID tags. Second, we create a spatiotemporal trajectory to trace the tags. Third, we improve the query performance by combining interrupted and localized tag trajectories into a complete trajectory. Therefore, TR-tree is a novel tag trajectory index that manages tags' movements and updates in RFID systems.

2. Previous Work

We previously reviewed the Time-Parameterized Interval R-tree (TPIR-tree) [2], which is one of the few indexing schemes that are capable of managing tag trajectories in RFID systems. The time-parameterized interval, or tp-interval, is dependent on the time in a reader. In TPIR-tree,

if a tag enters and does not leave a reader, the tag's time trajectory is presented as a point and no tp-interval is created. We call this a '*point tag*' τ_p. If there is no tp-interval, the tag can't be searched in the index.

TPIR-tree dictates that the tp-interval has only an entry time and not an exit time in the tag trajectory. The tp-interval has the form $\langle t_{id}, x, y, t_{enter}, now\rangle$. The *now* element, which is the present time, is called now-interval and is denoted '*nowI*' [2]. In TPIR-tree, if a tag is a point tag, the time of the tag must be extended to *now* when processing queries. The augmentation of the time-dimension results in the expansion of MBB [2]. From what has been outlined above, we can search a tag that possesses only an entry time in TPIR-tree.

TPIR-tree has three shortcomings. First, tag trajectories exist within each reader but not for intervals during which the tag is moving from one reader to another. Second, since MBB should be augmented for query processing, MBBs overlap. This degrades search performance and increases CPU time. Third, tag trajectories exist independently within each reader, so they cannot solve the problem of time intervals between trajectories. Hence, we focused on the construction of a tag index that combines localized trajectories.

3. RFID Tag Trajectory

Characteristics of Tag Trajectories. The trajectory of a tag is constructed by linking the local trajectories at various readers. The time trajectory appears along with the time axis in each reader. To construct a tag trajectory in practice, we define the trajectory as follows. Square brackets ($[t_e, t_l]$) are used to denote closed time intervals.

Definition 1: (*The time duration that a tag stays in a reader.*) Let t be the time when a tag enters the interrogation area of a reader r_i, and e and l be the events corresponding to the tag's arrival and departure from the interrogation area. The duration, t_d, in which the tag stays in the reader is as follows:

$$r_i.t_d = \{[t_e, t_l] \mid t_e \le t_d \le t_l\} \ . \tag{1}$$

Definition 2: (*Tag trajectory in a reader.*) Let $r.x$ and $r.y$ be the x- and y-coordinates of the reader, respectively. The tag trajectory T_r in a reader is expressed as follows:

$$T_r = \{(x, y, t_d)_r \mid x = r.x, y = r.y, t_d = [t_e, t_l]\} \ . \tag{2}$$

The t_d is expressed as t in the paper. Like a moving object's trajectory, a tag trajectory is expressed as a line from $(x, y, t)r_i$ to $(x, y, t)r_j$, where r_j is the reader that the tag passes after reader r_i. However, there are intervals during which the tag is out of range of any reader. To improve the efficiency of tag tracing, trajectories created over time by different readers can be connected to form a continuous line. The interruptions in the time trajectory are defined as *tagless time intervals* in Definition 3.

Definition 3: (*Tagless time interval (TTI).*) Let r_i and r_j be two readers passed by tag τ. The periods of time that τ stays in the readers are $[t_e, t_l]r_i$ and $[t_e, t_l]r_j$, respectively. There is a time interruption between readers r_i and r_j. This interval is defined as follows:

$$TTI = \begin{cases} r_j.t_e - r_i.t_l & ,(i < j) \\ 0 & ,otherwise \ . \end{cases} \tag{3}$$

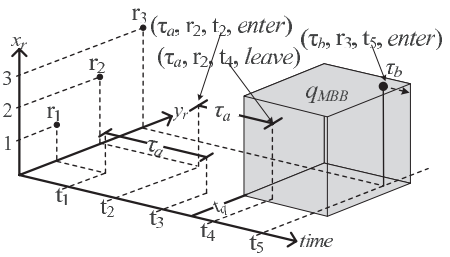


Fig. 2 3-dimensional expression of tag trajectory and point tag.

In Fig. 2, the tag trajectory is presented by three-dimensional form of location and time based on the readers. If a tag that has entered reader r_3 does not leave it, the tag trajectory is expressed as point τ_b in the index. This point does not have a trajectory, so it is not detected by any query. The query q_{MBB}, which overlaps with the time $[t_2, t_4]$ of tag τ_a, returns tag τ_a at query time t_q.

Trajectory Representation of Tags. A tag's moving path is connected to construct the full trajectory. The time trajectory in a reader is based on the enter and leave events $[t_e, t_l]$, where $t_e < t_l$. If the interrogation areas of two readers do not overlap, the relationship of the time trajectory between reader r_i and r_j is given by

$$(t_e, t_l)r_i \le (t_e, t_l)r_j \qquad (i = 1, 2, \ldots, n, \ \ i < j) \ . \tag{4}$$

If the interrogation area of two readers does overlap, the created time trajectories also overlap. A tag τ establishes the following relationship between reader r_i and r_j:

$$r_i.t_e < r_j.t_e < r_i.t_l < r_j.t_l \ . \tag{5}$$

Definition 4: (*Tag's spatio-temporal trajectory.*) Let Δt be the length of a trajectory interruption interval, *TTI*. The *TTI* is $\Delta t = r_j.t_e - r_i.t_l$ $(i < j)$. The spatiotemporal trajectory is expressed as an augmented data structure, $\langle x, y, [\Delta t, t_e, t_l]\rangle$.

The time trajectory, $[\Delta t, t_e, t_l]$, has the following properties.

Property 1. *The Δt and t_l of the first tag are null.* If a tag visits a reader for the first time and stays there, the time trajectory is expressed as $[\phi, t_e, \phi]$. This tag is expressed as a point tag that has only a location and an entry time rather than a trajectory. In order to process queries, t_l is assigned as the maximum time of the corresponding MBB to which the tag belongs next:

$$\tau_p.t_l = MAX\_TIME(MBB) \ . \tag{6}$$

Property 2. *A tag that visits only one reader has a null Δt.* If a tag leaves a reader, it has a time trajectory of $[\phi, t_e, t_l]$

until it enters another reader.

Property 3. *A tag that visits one reader and then enters another reader has a* Δt. *i.e.*, the time difference between leaving the previous reader and entering the current one, which is non-null. A tag that passes multiple readers has non-null values of Δt, which are ($pre\_reader.t_l - cur\_reader.t_e$). Thus, the time trajectory is $[\Delta t, t_e, t_l]$. In particular, t_l is null when the tag visits multiple readers and enters another reader.

4. Tag Trajectory Index in RFID Systems

Index Structure. An entry at a leaf node is composed of $\langle MBB, \tau_{id}, object\_pointer \rangle$. An MBB is a 3-D interval for the tag τ_{id} and consists of (x, y, t). Here, x and y are reader's locations comprising the spatial trajectory of the tag. The value t is the time that the tag spends in the readers and is represented in the form $[\Delta t, t_e, t_l]$. The real location of a tag τ_{id} stored in the database is pointed to by the *object_pointer*. A non-leaf node entry has the form of $\langle MBB, child\_pointer \rangle$. The size of a node is 4 KB, the block unit for disk pages.

In Fig. 3, TL contains all of the tags. An entry in TL has a leaf node pointer that links the tag identifications (IDs) to the TR-tree. We access the leaf nodes directly using TL. One advantage of TL is that the cost of dynamic updates of tags is low. In TR-tree, a tag's time information is updated as follows. For example, as shown in Fig. 2, if tag τ_a leaves a reader r_1 at time t_3, first the tag is found in TL and then the time property of the leaf node is changed to $[\phi, t_1, t_3]$ and written to the index. If this process is executed in R-tree, which has three levels as in Fig. 3, the nodes have to be accessed at least four times, whereas in TR-tree, two accesses are sufficient. In this paper, however, TL is a minor subject. We focus on tag trajectory and query processing.

Tag Search Algorithm. In TR-tree, tag search is divided into four query types, as follows. Algorithm 1 is a basic search algorithm reflecting these query types.

- *OBJECT* query: $q = (t_{id})$, returns a tag of t_{id} with its current location and time.
- *TIME* query: $q = ([t_{start}, t_{end}])$, returns all the tags within the designated time interval.
- *TRAJECTORY* query: $q = (t_{id}$, time/location), returns a time or location trajectory for the tag.
- *SCOPE* query: $q = (x_1, x_2, y_1, y_2, [t_{start}, t_{end}])$, returns the tag corresponding to the time interval within the designated spatial scope. At that time, if the spatial coordinates are not specified, they are the same as the *TIME* query.

5. Performance Evaluation

Experimental Environments. We compared R-tree, TPIR-tree and TR-tree by implementing the environment in C++. R-tree was modified to store the time dimension. The experiments were conducted on a Pentium IV 2.6-GHz computer with 1 GB of memory running Windows XP. The simulation data were created using the *Oporto* [5] generator for moving tags, with datapoints having the structure $\langle tag\_id, (x_r, y_r), [enter\_time, leave\_time] \rangle$. The movement of the tags is assumed to be uniform. Table 1 shows the experimental parameters. The time intervals in the dataset are composed of the closed time interval $[t_1, t_2]$ and the open time interval $[t_1, t_2)$.

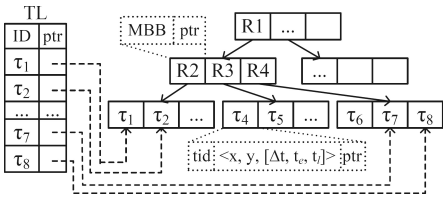


Fig. 3 TR-tree structure with TL.

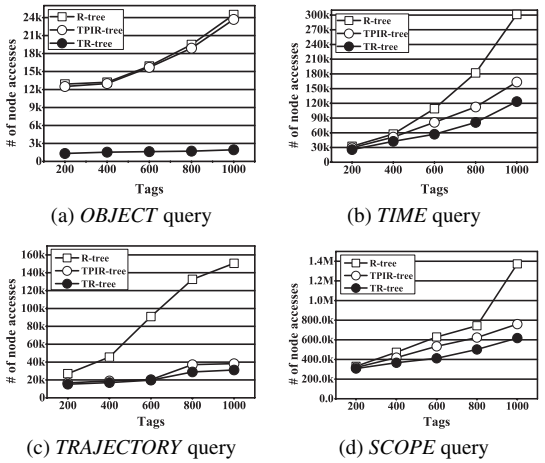
Algorithm 1 Tag search: Search tags by a query

Procedure TagSearch (*tagID, tagLocation, tagTimeInterval*)
Input: *tagID* is a tag identification, *tagLocation* is tag location, *tagTimeInterval* is time of a tag
Output: Searched tags or a trajectory
Begin
1: Read the ROOT node of TR-tree;
2: **Select Case** *option*
3: **Case** Object(*tagID*):
4: Seek the *tagID* in TL index;
5: **if** *tagID* exists in TL **then return** an object;
7: **Case** Time(*tagTimeInterval*):
8: **do** Search tags between starting and end time;
9: tagObjectList += object **Loop**;
10: **return** tagObjectList;
11: **Case** Trajectory(*tagID*,[*tagLocation*/*tagTimeInterval*]):
12: Seek the *tagID* in TL index and access to an entry;
13: **return** a trajectory in location or time;
14: **Case** Scope(*tagLocation*, *tagTimeInterval*):
15: **do** Traverse TR-tree with location [x1, x2], [y1, y2] and time interval $[t_s, t_e]$;
16: **if** found an object **then** tagObjectList += object **Loop**;
17: **return** tagObjectList; //query result
18: **End Select**
End

Table 1 Simulation parameters.

Item	Contents	Remarks
time interval	$[t_1, t_2]$ or $[t_1, t_2)$	closed/open
tags	1000 tag data	synthetic
point tags ratio	30, 60, 90%	not leaved
query workload	100 query lines	each query

Node Accesses of Queries. The performance of R-tree variants depends on the number of node accesses. The node accesses mainly involve insert, update, and search queries. These operations increase the amount of disk i/o. Figure 4 (a) compares the average number of node accesses when searching for a tag in TR-tree, TPIR-tree, and R-tree using *OBJECT* query. The results were obtained by executing 500 queries. Using TR-tree, the results for 200 to 1000 tags were homogenous because the leaf nodes are accessed directly. However, the number of node accesses in TR-tree increases slightly with the number of tags because there is



(a) *OBJECT* query (b) *TIME* query

(c) *TRAJECTORY* query (d) *SCOPE* query

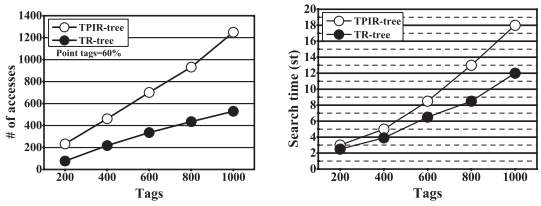
Fig. 4 The number of node accesses with number of tags in query types.

some backtracking caused by overlapping MBBs.

TIME query searches tags within a time window. Figure 4 (b) shows the results for returning all the tags in a specific time window. In TPIR-tree, since a point tag's time is extended to the present time, MBBs overlap. The performance of TPIR-tree is worse than that of TR-tree. *TRAJECTORY* query results (Fig. 4 (c)) of TPIR-tree are similar to those of TR-tree. However, R-tree produces very high numbers because it uses a different structure to store trajectories. In Fig. 4 (d), *SCOPE* query has high numbers from the start because it needs three parameters, two points of an MBR and a time window. Since there is expansion of MBBs in TPIR-tree, this yields a high number of node accesses.

Update Cost and Search Time. We simulated the update cost for 30, 60 and 90% of the point tags. However, as space is limited, we show only Fig. 5 (a). In Fig. 5 (a), the number of node accesses increases when the number of tags increases. TPIR-tree shows a much larger number of accesses because it searches for tags to be updated in top-down manner [4]. Another reason for the poor performance is that if a point tag (*nowEntry* in TPIR-tree) is encountered in a query, MBB is expanded to the present time [2] for the trajectory. If an MBB is expanded, it overlaps with other MBBs, which in turn causes a backtracking and increases the number of node accesses. In comparison, TR-tree already has *MAX_TIME* for the point tag's trajectory. That is, since no expansion of MBB occurs in TR-tree, the number of node accesses is much smaller.

We simulated the search time of queries using CSIM simulator [6]. Since *TIME* query returns all the tags within the designated time interval, it is fit to check the performance of the time-based query by CSIM. The simulator generates the simulation time (st), which is independent of the operating system.



(a) Update cost of tags (b) Search time of tags

Fig. 5 Update cost according to a point tag ratio of 60% and a search time via *TIME* query (st: simulation time).

In Fig. 5 (b), since the time dimension of TPIR-tree is extended to the present time, the search time is higher than that of TR-tree. That is, the result of Fig. 5 (b) is that TR-tree decreases the number of overlapping MBBs compared with TPIR-tree.

6. Conclusion

Fast tag tracing is essential in RFID systems. We propose a novel tag trajectory index that solves the problem of localized tag trajectories between readers, as well as the problem of point tags being observed. Since no augmentation of MBBs is required in TR-tree, MBBs do not overlap in the query time. Therefore, TR-tree provides a considerable improvement in node accesse, update and search time performance. As shown in our simulation, the performance of TR-tree is superior to existing indices. The number of node accesses decreases more gradually as the point tag ratio and the number of tags increases.

References

[1] H. Vogt, "Efficient object identification with passive RFID tags," LNCS2414, pp.98–113, Springer, 2002.
[2] C.H. Ban, B.H. Hong, and D.H. Kim, "Time parameterized interval R-tree for tracing tags in RFID systems," LNCS 3588, pp.503–513, Springer, 2005.
[3] A. Guttman, "R-trees: A dynamic index structure for spatial searching," Proc. ACM SIGMOD conference on Management of data, vol.14, pp.47–57, 1984.
[4] D. Kwon, S. Lee, and S. Lee, "Indexing the current positions of moving objects using the lazy update R-tree," Proc. Third Intl. Conference on Mobile Data Management, pp.113–120, 2002.
[5] J.-M. Saglio and J. Moreira, "Oporto: A realistic scenario generator for moving objects," Workshop on Database and Expert Systems Applications, pp.426–432, 1999.
[6] H. Schwetman, "CSIM19: A powerful tool for building system models," Proc. 2001 Winter Simulation Conference, pp.250–255, 2001.
[7] K.C.K. Lee, J. Schiffman, B. Zheng, W.-C. Lee, and H.V. Leong, "Tracking nearest surrounders in moving object environments," Proc. Pervasive Services, 2006 ACS/IEEE International Conference, pp.3–12, 2006.