Efficient Enforcement of Security Policies based on Tracking of Mobile Users*

Vijayalakshmi Atluri and Heechang Shin

MSIS Department and CIMIC, Rutgers University, USA
{atluri,hshin}@cimic.rutgers.edu

Abstract. Recent advances to mobile communication, Global Positioning System (GPS) and Radio Frequency Identification (RFID) technologies have propelled the growth of a number of mobile services. These require maintaining mobile object's location information and efficiently serving access requests on the *past, present* and *future* status of the moving objects. Moreover, these services raise a number of security and privacy challenges. To address this, security policies are specified to ensure controlled access to the mobile user's location and movement trajectories, their profile information, and stationary resources based on the mobile user's spatiotemporal information. Considering the basic authorization specification $\langle subject, object, privilege \rangle$, in a mobile environment, a moving object can be a subject, an object, or both. Serving an access request requires to search for the desired moving objects that satisfy the query, as well as enforce the security policies.

Often, enforcing security incurs overhead, and as a result may degrade the performance of a system. To alleviate this problem, recently Atluri and Guo have proposed an unified index structure, ^{S}TPR-tree, to organize both the moving objects and authorizations specified over them. However, the ^{S}TPR-tree is not capable supporting security policies based on *tracking* of mobile users. In this paper, we present an index structure, called S^{PPF}-tree, which maintains past, present and future positions of the moving objects along with authorizations by employing *partial persistent storage*. We demonstrate how the S^{PPF}-tree can be constructed and maintained, and provide algorithms to process two types of access requests, including moving object requests by stationary subjects such as *locate* and *track*, and stationary object requests by moving subjects.

1 Introduction

Recent advances to mobile communication, Global Positioning System (GPS) and Radio Frequency Identification (RFID) technologies have propelled the growth of a number of mobile services. Location-based service is one such example, which aims at delivering personalized services to mobile customers. These include: providing nearby points of interest based on the real-time location of the mobile

* This work is supported in part by the National Science Foundation under grant IIS-0242415.

customer, advising of current conditions such as traffic and weather, deliver personalized, location-aware, and context-sensitive advertising based on mobile customer profiles and preferences, or provide routing and tracking information. Delivery of these services requires maintaining mobile object's location information as well as the preference profiles of the customers carrying these mobile objects. In addition, it requires efficient processing of access requests to find the *past, present* and *future* status of the moving objects.

Since effective delivery of a mobile service may need to locate and track a mobile customer, and gain access to his/her profile, it raises a number of security and privacy challenges. Location information has the potential to allow an adversary to physically locate a person. As such, wireless subscribers carrying mobile devices have legitimate concerns about their personal safety, if such information should fall into the wrong hands. Moreover, services such as targeted advertising may deliver the service based on the mobile customers' profile and preferences. As such, privacy of mobile users can be compromised if the sensitive profile information of the mobile users is revealed to unintended users. Therefore, it is important that the sensitive profile information is revealed only on the need-to-know basis. In addition to the privacy concerns mentioned above, there are a number of applications that call for securing resources based on the criteria of mobile objects. These include context (location)-sensitive access control, and ubiquitous computing environment, where access is permitted based on the location of the subjects/objects during a specific time.

In summary, in a mobile environment, there are a number of applications that require enforcing security policies to provide controlled access to the mobile user profiles, to their current location and movement trajectories, to mobile resources, stationary resources based on the user's spatiotemporal information. Thus, an appropriate access control mechanism must be in place to enforce the authorization specifications reflecting the above security and privacy needs.

Traditionally, access policies are specified as a set of authorizations, where each authorization states if a given subject possesses privileges to access an object. Considering the basic authorization specification $\langle subject, object, privilege \rangle$, in a mobile environment, a moving object can be a subject, an object, or both. Access requests in such an environment can typically be on *past, present* and *future* status of the moving objects [1, 2]. Serving an access request requires to search for the desired moving objects that satisfy the query, as well as enforce the security policies.

Often, enforcing security incurs overhead, and as a result may degrade the performance of a system. One way to alleviate this problem and to effectively serve access requests, is to efficiently organize the mobile objects as well as authorizations. Towards this end, recently Atluri and Guo [3] have proposed a unified index structure called ^{S}TPR-tree in which authorizations are carefully overlaid on a moving object index structure (TPR-tree), based on their spatiotemporal parameters. One main limitation of the ^{S}TPR-tree is that it is not capable of maintaining past information. As a result, it cannot support queries based on past location and security policies based on *tracking* of mobile users.

In this paper, we present an index structure, called S^{PPF}-tree, which maintains past, present and future positions of the moving objects along with authorizations by employing the *partial persistent storage*. In particular, we build on the concepts of the R^{PPF}-tree [4] and overlay authorizations suitably on the nodes of the index tree. Essentially, a partial persistent structure keeps all past states of the data being indexed, but updates only the newest version. We demonstrate how the S^{PPF}-tree can be constructed and maintained, and provide algorithms to process access requests. Specifically, we support two types of access requests: requests for moving objects by stationary subjects and requests for stationary objects by moving subjects. The first type of requests, in addition to retrieving the moving objects in a specific spatiotemporal region, allow retrieving the location of the moving objects as well as their trajectories. As such our model would support security policies based on tracking of moving objects.

This paper is organized as follows. In section 2, we present our moving object authorization model. We present the preliminaries in section 3. In section 4, we present our proposed novel unified index structure, the S^{PPF}-tree and illustrate our approach and strategy to overlay authorizations on top of the R^{PPF}-tree. In this section, we also describe how access requests are evaluated. Related work is presented in section 5. In section 6, we conclude the paper by providing some insight into our future research in this area.

2 Moving Object Authorization Model

In this section, we introduce an authorization model for moving object data, which is an extension of the model proposed in [3]. In a moving object environment, authorization specifications should be capable of expressing access control policies based on spatiotemporal attributes of both subjects and objects.

Definition 1 (Authorization). *An authorization α is a 4 tuple $\langle ce, ge, p, \tau \rangle$, where ce is a credential expression denoting a set of subjects, ge is a object expression denoting a set of objects, p is a set of privilege modes, and τ is a temporal term.*

The formalism to specify ce, ge and τ has been developed in [5]. Credential expression ce can be used to specify a set of subjects such that they are associated with (i) a set of spatiotemporal and/or other traditional credential attributes, (ii) a set of subject identifiers, or (iii) a combination of both. In the same way, object expression ge can be used to specify a set of objects such that they (i) are associated with a set of spatiotemporal and/or other types of attributes, (ii) a set of object identifiers, or (iii) a combination of both. Note that the set of subjects and objects denoted by ce and ge can be moving objects. To avoid confusion, from now on, we denote the objects specified in the authorization as *auth-objects* (stands for authorization objects). τ can be a time point, a time interval or a set of time intervals.

Our model supports not only *read*, *write*, and *execute* privileges for traditional auth-objects but also *viewing* and *compose* for moving objects. We support three viewing privileges: *View, Locate,* and *Track* privileges allow subjects

to access a moving object(s), to read the location or trajectory information of a moving object(s) in the authorized spatiotemporal region, respectively. Compose privileges allow subjects to write information on the auth-objects. In the following, we present some examples of security policies.

- **Policy 1:** A mobile customer is willing to reveal his personal profile information to a merchant only during the evening hours, and while he is close to the shopping mall. In this case, only the auth-object (customer) is a moving object and this policy is based on auth-object's spatiotemporal attributes.
- **Policy 2:** An employee is allowed to update certain company data only during "office hours" and while "in the office." Note that only the subject (employee) is a moving object. Also note that the policy is based on the subject's spatiotemporal attributes.
- **Policy 3:** An airport security official can access the trajectory information of travelers in the airport only while he is on-duty (i.e., during 11pm-7am). In this case, both the subject and the auth-objects are moving objects, and the policy is based on the spatiotemporal attributes of both subject and auth-object.
- **Policy 4:** A FBI agent can access the current location and trajectory information of a truck with id 325. Note that although the subject and the auth-object are moving objects, the subject is allowed to access the information regardless of his location and time. In this case, the policy is based on the identifiers of both subject (FBI agent) and auth-object (truck with id 325).
- **Policy 5:** A police office in Newark, NJ can access only the dispatched patrol cars from the Newark area. Note that only auth-objects are moving objects. Also, the policy is specified on two types of auth-objects: object identifiers (patrol cars from Newark police station) and spatiotemporal region.

The above policies can be specified as the following authorizations.

- α_1=⟨merchant(i), {profile(i)∧rectangle(j)=(5,6,1,2)∧[5pm,9pm]},locate⟩
- α_2=⟨{emp(i)∧ rectangle(j)=(3,5,1,5)∧[9am,5pm]},{profile(j)}, update ⟩
- α_3=⟨{security_official(i) ∧ rectangle(j)=(1,4,3,4) ∧ [9am, 5pm]}, {travelers(i) ∧ rectangle(j)=(100,50,30,30) ∧ [current time]}, track ⟩
- α_4=⟨ FBI_agent(i), truckid(j)=325, track ⟩
- α_5=⟨{dispatch_department(i) ∧ office_location(j)='Newark'}, {patrol_cars (k)∧rectangle(l)=(10,50,30,30) ∧ dispatched_from(k) = 'Newark'}, track ⟩

3 Preliminaries

In this section, we present the partial persistence framework and review the R^{PPF}-tree [4], a moving object index that maintains not only the *present* and anticipated *future* positions of moving objects, but also their *past* positions.

Representation of Moving Objects: Let the set of moving objects be O=$\{o_1, \ldots, o_n\}$. In the d-dimensional space, objects are specified as points which move with constant velocity $\bar{v} = \{v_1, v_2, \ldots, v_d\}$ and initial location

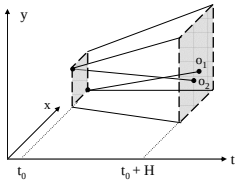

Fig. 1. The Time Parameterized Bounding Rectangle (*tpbr*)

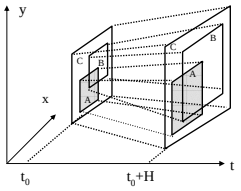

Fig. 2. The *tpbr* Hierarchy

$\bar{x} = \{x_1, x_2, \ldots, x_d\}$. The position $\bar{x}(t)$ of an object at time $t(t \geq t_0)$ can be computed through the linear function of time, $\bar{x}(t) = \bar{x}(t_0) + \bar{v}(t - t_0)$ where t_0 is the initial time, and $\bar{x}(t_0)$ the initial position. Considering a two-dimensional space, a moving object o_i moving in $\langle x, y \rangle$ space can be represented as $o_i = ((x_i, v_{i_x}), (y_i, v_{i_y}))$.

Time Parameterized Bounding Rectangle (*tpbr*): Given a set of moving objects $O = \{o_1, \ldots, o_n\}$ in the time interval $[t_0, t_0 + \delta t]$ in $\langle x, y, t \rangle$ space, the *tpbr* of O is a 3-dimensional bounding trapezoid which bounds all the moving objects in O during the entire time interval $[t_0, t_0 + \delta t]$: $tpbr(O) = \{(x^{\vdash}, x^{\dashv}, y^{\vdash}, y^{\dashv}), (v_x^{\vdash}, v_x^{\dashv}, v_y^{\vdash}, v_y^{\dashv})\}$ where $\forall\, i \in \{1, 2, \ldots, n\}$, $x^{\vdash} = min_i\{x_i(t_0)\}$, $x^{\dashv} = max_i\{x_i(t_0)\}$, $y^{\vdash} = min_i\{y_i(t_0)\}$, $y^{\dashv} = max_i\{y_i(t_0)\}$, $v_x^{\vdash} = min_i\{v_{i_x}\}$, $v_x^{\dashv} = max_i\{v_{i_x}\}$, $v_y^{\vdash} = min_i\{v_{i_y}\}$, $v_y^{\dashv} = max_i\{v_{i_y}\}$.

Time Horizon (H): Given a moving object, it is unrealistic to assume that its velocity remains constant. Therefore, the predicted future location of a object specified as a linear function of time becomes less and less accurate as time elapses [6]. To address this issue, a *time horizon H* is defined, which represents the time interval during which the velocities of the moving objects assumed to be the same. Figure 1 shows how *tpbr* bounds the trajectory of two moving objects o_1 and o_2 in $[t_0, t_0 + H]$.

The Tree Structure: Given a set of *tpbr*s, they can be organized in a hierarchical structure. In figure 2, *tpbr* C encloses *tpbr*s A and B. These three can be organized as a hierarchical structure with A and B being the children of C. Essentially, at the bottom-most level of the hierarchy, a set of moving objects could be grouped to form *tpbr*s. Each *tpbr* of the next higher level is the bounding *tpbr* of the set of *tpbr*s of all of its children. The root of the hierarchy is thus the bounding *tpbr* covering all its lower level *tpbr*s in a recursive manner.

The Partial Persistence Framework: Partial persistence is a data structure that keeps all past states of the data being indexed, but applies updates only to the newest version. It is based on the following important concepts.

- **Evolution of Index Nodes and Data Entry:** In order to be transformed to a partially persistent structure, each index (leaf or index) node and data entry (moving object) include two additional fields for maintaining the evolution of the index records: *insertion time* and *deletion time*. These are denoted as $N.insertionTime$ and $N.deletionTime$ for node N. If a new moving ob-

ject is available and captured at time t_0, its insertion time is set to t_0 and deletion time is set to ∞. When the object is logically deleted from the index at time t_d, its deletion time is changed from ∞ to t_d. The same rule applies to index nodes. A node or a data entry is said to be *dead* if its deletion time is less than ∞, otherwise it is said to be *alive*.

- **Time Split:** When an update (insertion or deletion) occurs at a node N, it may result in structural changes if it becomes underfull or overfull. If this is the case, a *time-split* occurs to N. The time-split on N at time t is performed by copying all alive entries in N at t to a new leaf node L and timestamp of both L and those copied entries are set to $[t, \infty)$. In addition, the deletion time of N is set to t, and N is considered dead. Then, the new node L is investigated further in order to incorporate it into the tree. Essentially, three different cases may arise: (i) split: If L is overfull, split it into two nodes and then insert these two nodes into the tree. (ii) merge: If L is underfull, accommodate by merging it with another node. (iii) no change: If L is neither overfull or underfull, insert it directly into the tree. After the structural change, the *tpbr* of the parent node may need to be updated accordingly and the described process may be repeated up to the root node. If the root node is time-split at time t, a pointer to the new alive node together with timestamp $[t, \infty)$ is added to a special root array that is stored in the main memory [4].

  Note that if the tree is constructed at t_0 and time split for the alive root element of the root array occurs at $\{t_1, t_2, \ldots, t_n\}$, each root element in the root array is associated with time interval $[t_0, t_1), [t_1, t_2), \ldots, [t_{n-1}, t_n)$, and $[t_n, \infty)$. The associated time interval for each root element represents the valid structure of the tree during those time intervals. Thus, if we want to know the status of the tree at time t, we simply need to find a root element r from the root array such that the time interval of r includes t.

In the following, we explain the concept of time-split, root array, dead and alive nodes by taking a concrete example. Consider a tree with a node that can hold 5 data entries. Obviously, the node is considered underfull if the number of data entries is less than 2, and overfull if the number of data entries is more than 5. Observe that the dead nodes are shaded in figures.

- **Time interval t = [0, 4]:** Moving objects o_1, o_2, and o_3 are inserted into the root node at t=0: the insertion time and deletion time of all these objects are set to [0,∞). Then at $t = 2$ and 3, o_4 and o_5 are inserted, and, thus, their insertion time and deletion times are [2,∞) and [3,∞), respectively. At $t = 4$, o_6 is inserted to the root node N, which becomes overfull. So, time split occurs. A new leaf node L with insertion and deletion time $[4, \infty)$ is created and all the alive data entries $(o_1, \ldots, o_6)$ in the root node are copied there with insertion and deletion times as $[4, \infty)$. Because L is also overfull, it is split into two nodes, which are inserted into the tree. A new root entry is added, forming a root array. The previous root's deletion time is set to 4, representing it as a dead node, and the time interval of the newly created root is set to [4,∞), as shown in Figure 3.

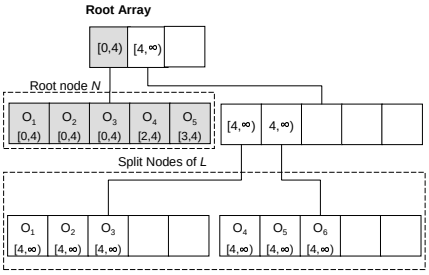

Fig. 3. The Index Structure at time 4

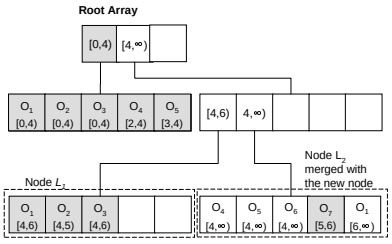

Fig. 4. The Index Structure at time 6

– **Time interval t = [5, 6]:** At $t = 5$, o_2 is deleted and o_7 is inserted. Thus, the deletion time of o_2 is set to 5, and o_7 is inserted in the tree with the insertion and deletion times $[5, \infty)$. Then, at $t = 6$, o_3 and o_7 are deleted. So, deletion time of these objects are set to 6. Because the deletion of o_3 results in the underfull of the node L_1 that stores o_3, a time split occurs: another new node K is created and alive entry o_1 is copied there. Since newly created node K is underfull, it is merged with its neighboring alive node L_2. The deletion time of the node L_1 is set to 6, representing that L_1 is dead. The resultant data structure is shown in figure 4.

When update occurs, the resulting trajectory of a moving object may consist of disconnected and slightly incorrect segments because at the insertion of the object, the predicted future positions can be different from the actual positions. Therefore, during update, the last-recorded trajectory segment of an object needs to be updated. It may be stored in more than one leaf node because the leaf node in question may have been time split a number of times since the previous updates [4]. R^{PPF}-tree corrects the last-recorded trajectory segment by visiting all leaf nodes that contain copies of the segment and also tightens the *tpbr* accordingly. For example, in figure 4, suppose the actual location of o_3 turns out to be different from the predicted location during update (deletion). Then, after setting the deletion time of o_3 as 6, all the nodes that include the trajectory of o_3 since the last update (insertion of o_3 at $t = 3$) are updated to point the actual location of o_3 correctly. The first root element and the node L_1 is such a case.

4 The S^{PPF}-Tree

In this section, we present our proposed unified index, the S^{PPF}-tree that indexes authorizations as well as the moving objects by capturing their past, present, and future locations. As a result, we can support authorizations based on *locate* and *track* privileges.

4.1 Authorization Overlaying

Our approach is to first construct a R^{PPF}-tree index for moving objects, and then overlay authorizations on top of each node of the index by carefully exam-

ining the spatiotemporal extents of both the node and the authorizations. The resulting tree is the S^{PPF}-tree. We denote the spatiotemporal extent of an authorization α by $\alpha^{\square}$. In other words, x-axis interval = $[\alpha.x_b, \alpha.x_e]$, y-axis interval = $[\alpha.y_b, \alpha.y_e]$, and t-axis interval = $[\alpha.\tau_b, \alpha.\tau_e]$ where $\alpha.x_b$, $\alpha.x_e$, $\alpha.y_b$ and $\alpha.y_e$ denote the spatiotemporal extent specified by *ce* or *ge* represented by the lower and upper bounds in the x and y axes, respectively, and $[\alpha.\tau_b, \alpha.\tau_e]$ denote the time interval during which α is valid. Also, we denote the spatiotemporal extent (*tpbr*) of a node N by $N^{\square}$.

An authorization α is said to be *subject-based authorization*, if $\alpha^{\square}$ is computed from $\alpha.ce$. Similarly, if $\alpha^{\square}$ is from $\alpha.ge$, it is said to be an *object-based authorization*. In other words, in the former case, subjects are the moving objects and in the latter case, objects are the moving objects. In our tree, we are capable of overlaying only if the authorization is specified based on the spatiotemporal extent of *ce* or *ge*, but not both. In addition, we assume that $\alpha^{\square}$ is a contiguous spatiotemporal region without losing any generality because each non-continuous spatiotemporal region can be sliced to form a single contiguous region.

A node of S^{PPF}-tree is similar to that of R^{PPF}-tree except that it includes two pointers that point to a set of subject-based authorizations (α_S) and a set of object-based authorizations (α_O) overlaid on the node.

The overlaying strategy first selects the root nodes from the root array such that the root node's alive time interval is overlapped with the authorization's time interval. Then, for each selected root node r, it traverses the tree recursively starting from the root node r to the leaf level in a way that for each node N in the traversal path, $\alpha^{\square}$ is compared with $N^{\square}$. All the possible scenarios for this comparison are as follows:

- **Case 1:** If the spatiotemporal extent of α fully encloses that of the node N, we will stop traversing and overlay α on N. This is because, if a subject is allowed to access objects within a certain spatiotemporal region, it is allowed to access objects in the *subregion* of that [3]. (If $\alpha.ge$ points to a set of auth-objects instead of a spatiotemporal region, we can exclude unauthorized auth-objects by post-processing the query result when we evaluate the query.) After overlaying an authorization on a node, it is not necessary to overlay the same authorization on any of its descendents.
- **Case 2:** If the spatiotemporal extent of α overlaps with that of the node N, the level of the node decides where it is overlaid.
  - If N is a non-leaf node, each of N's children is traversed and the algorithm repeats the comparison between $\alpha^{\square}$ and each child$^{\square}$. The goal here is to check if there exist a child of N whose spatiotemporal extent is enlosed by that of α.
  - If the node N is a leaf node, we overlay α on the leaf node N. This is because, when the spatiotemporal extent of the authorization $\alpha^{\square}$ does not enclose, but overlaps with that of the leaf node $N^{\square}$, we need to ensure that no relevant authorizations are discarded. Also, note that only part of the spatiotemporal extent of $N^{\square}$ is in the authorized region. The

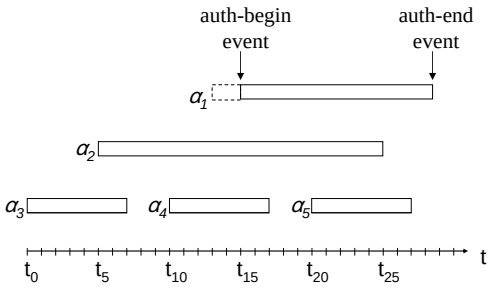

Fig. 5. Authorization Time Line

moving objects from the remaining unauthorized spatiotemporal region $N^{\square} - \alpha^{\square}$ must be removed from the user's output, if the user request includes this region.

– **Case 3:** Else, which implies the spatiotemporal extent of the authorization, $\alpha^{\square}$ is disjoint with that of the node $N^{\square}$, we stop the overlaying process. This is because, if $\alpha.ce$ does not have privilege to the region covered by $N^{\square}$, then α is not applicable to that region. Also, since $N^{\square}$ includes spatiotemporal extent of all of its children nodes, $\alpha^{\square}$ is disjoint with the spatiotemporal extent of each child. Thus, there is no need to traverse further to the leaf level.

4.2 Maintenance of the S^{PPF}-tree

One main challenge of the S^{PPF}-tree is to maintain the overlaid authorizations as the tree evolves. Changes to the S^{PPF}-tree are needed due to the following two reasons:

– **Updates to the moving object:** It is important to note that, while the spatiotemporal region of an overlaid authorization is static in nature, the *tpbr* of each node in the tree changes over time. Therefore, it is possible that certain overlaid authorizations may no longer satisfy the conditions. As a result, it may be necessary to reposition the existing overlaid authorization.
– **Change of applicable authorizations:** If the overlaid authorizations are valid only during a certain time interval, as time elapses, they are no longer applicable. Therefore, these need to be removed from the overlaid set. Also, certain new authorizations may become applicable, which need to be overlaid appropriately.

Because the S^{PPF}-tree is an unified index that maintains not only moving objects but also authorizations, we need to pay attention to how updates of one type can be performed without hampering the properties of the S^{PPF}-tree.

4.2.1 Handling Updates due to Change of Applicable Authorizations: To handle this issue, we introduce the notion of *Authorization Log*, described below.

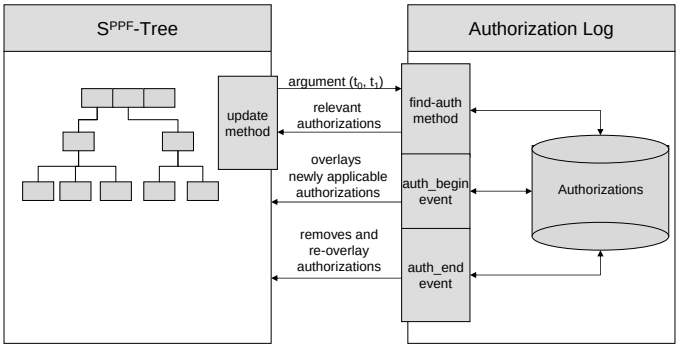

Fig. 6. Relationship of Authorization Log and S^{PPF}-Tree

Authorization Log: An authorization log is nothing but a data structure constructed by spreading all the authorizations on the time line. For each authorization, we consider the following two events: (1) **auth-begin** event and (2) **auth-end** event. These two are nothing but $[\tau.t_b, \tau.t_e]$ specified in the authorization specification. (Note that each authorization will have only two such events since we are not considering periodic authorizations. However, our proposed solution can be easily extended to handle periodic authorizations.) For example, in figure 5, the auth-begin event of the authorization α_1 occurs at time t_{15}, and the auth-end event will occur at time t_{28}.

Essentially, as time elapses, new authorizations may become applicable and we do not want to miss overlaying these authorizations on the S^{PPF}-tree. An authorization α is said to be applicable to the tree constructed at t, if the two time intervals $[\alpha.\tau_b, \alpha.\tau_e]$ and $[t, t+H]$ overlap. For example, suppose the S^{PPF}-tree is constructed at $t = t_{10}$, which is valid until $t_{10} + 2$ (assuming $H = 2$). Referring to figure 5, only α_2, α_3, and α_4 are overlaid on the tree. Since valid intervals of α_1 and α_5 are outside $[t_{10}, t_{10} + 2]$, they are applicable now and therefore are not overlaid on the tree. On the other hand, at t_{20}, both α_1 and α_5 must have been overlaid on the tree. However, the tree has no capability to keep track of newly applicable authorizations that need to be overlaid on the appropriate nodes of the tree. An auth-begin event triggers the OverlaySubtree procedure to take care of this issue. For example, α_1 in figure 5 will be overlaid on the tree at t_{13} because the tree is valid up to the current time + time horizon.

Also, after some time later, certain overlaid authorizations become invalid and therefore must be removed from the tree. This is taken care by the auth-end event to trigger such removals. The removed authorization needs to be re-overlaid on the S^{PPF}-tree because it may satisfy the overlaying conditions of another node in the tree.

In addition to triggering the overlaying and deletion of authorizations, update must take care of the cases when the time-split occurs. In this case, an entirely new node will be created for which there exist no overlaid authorizations. The

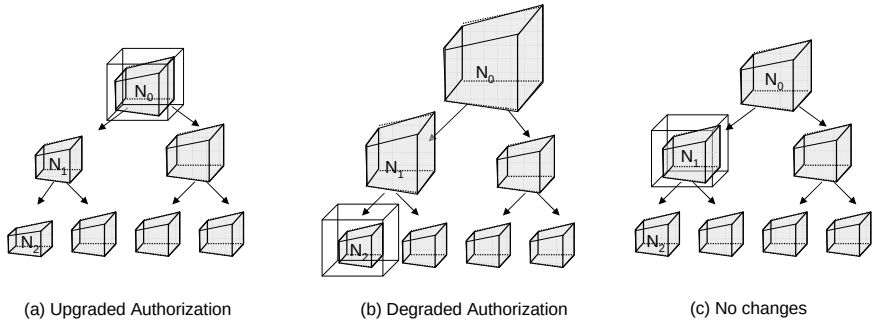

Fig. 7. Re-overlaying of authorizations due to updates to moving objects

find-auth method computes all the authorizations overlapping with the interval of the newly created nodes. Figure 6 depicts the relationship between the authorization log and the S^{PPF}-tree along with the auth-begin and auth-end events, and the find-auth method.

4.2.2 Handling Updates due to Changes to Moving Objects: Updates to moving objects may cause a structural change to the S^{PPF}-tree. When update (insertion/deletion) occurs on the S^{PPF}-tree, all the access nodes, which are ancestors of the leaf node which the update is applied to, need to be checked because the overlaid authorizations in the nodes may either be degraded authorizations (authorizations which were originally overlaid on the access nodes, but no longer fit in their original positions due to the spatiotemporal enlargement of the nodes by updates) or upgraded authorizations (authorizations which were originally overlaid on the access nodes, but able to fit in an ancestor of their original positions due to spatiotemporal shrinkage of the nodes by updates). This procedure is even more complicated if the update process results in the structural changes due to time-split. The details are summarized below.

1. **Update authorizations on the adjusted nodes:** Based on the periodic updates on the position of the moving objects, the *tpbr* of each node in the S^{PPF}-tree will be adjusted; they may either shrink or expand. Moreover, adjustments to the *tpbr* of a node may trigger adjustments to the *tpbr*s of its ancestor nodes. For each adjusted node N, every overlaid authorization α on it can fall into one of the three categories: (i) degraded authorization, (ii) upgraded authorization, (iii) no changes. The algorithm checks first if it is an upgraded authorization and attempts to overlay it as high in the tree as possible. Else, the same overlaying strategy is used to find the appropriate position for α. Figure 7 shows these three different cases. Suppose N_2 is the adjusted leaf node. Figure 7 (a) shows the shrinkage of the *tpbr* for N_2 and its parents. The authorization initially overlaid on N_1 is now repositioned to N_0: it can enclose $N_1^{\square}$ as well as $N_0^{\square}$ spatiotemporally and therefore becomes an upgraded authorization. On the other hand, the *tpbr* of N_2 may be expanded due to the adjustment. Figure 7 (b) shows this expanded case, and that the overlaid authorization does not enclose $N_1^{\square}$ spatiotemporally any more. It

becomes a downgraded authorization. Therefore, it is repositioned to the child of N_1, i.e., N_2. In addition, it may be possible that the shrinkage or expansion of the corrected node does not affect the overlaid authorizations if the overlaid authorization still encloses $N_1^{\square}$ but does not enclose its parent N_0 spatiotemporally. Figure 7 (c) presents this case.

2. **Overlay authorizations on the newly created node:** The newly created node due to a time-split does not have any authorizations overlaid on it. Therefore, all the authorizations whose valid time intervals are overlapped with the interval of this node are overlaid on the alive root node from the root array.

4.3 Access Request Evaluation

In this section, we present the different types of access requests and how these are evaluated against the specified authorizations to retrieve the information that satisfies the user request. Two types of user requests are possible under our framework.

- *Moving Object Request (MOR):* A subject who is stationary may wish to access moving objects that fall within a spatiotemporal region. This can be a *locate*, a *track* or a *view* request on a moving object.
- *Stationary Object Request (SOR):* A subject who is within a spatiotemporal region may wish to access objects that are stationary.

Definition 2 (Moving Object Request). *A* moving object request *(MOR), denoted as a triple $U = \langle s, Q, m \rangle$, where s is the subject of the user request, Q is a spatiotemporal region, and m is a track, a locate, or a view access mode.*

The result of a MOR would be one of the trajectory of an object(s), the position of an object(s), or the identifier of an object(s). A trajectory is of the form $\langle o, \{loc_1, loc_2 \ldots, loc_n\} \rangle$, where o is the object, and loc_i is the i^{th} location information of o in the x, y, t dimensional space. In case of locate, the result would be of the form $\langle o, loc \rangle$. The result of a view access mode would be a set of object ids. We use $U^{\square}$, $U.s$, $U.mode$, $U.\tau_b$, and $U.\tau_e$ to denote the spatiotemporal extent, the subject, the access mode, effective time interval $[U.\tau_b, U.\tau_e]$ of the access request U respectively.

Definition 3 (Stationary Object Request). *A* stationary object request *(SOR), denoted as a triple $V = \langle s, loc, o \rangle$, where s is the subject of the user request, loc is the current locations of the subject in the x, y, t dimensional space, and o is an auth-object that the subject s tries to gain access to.*

The spatiotemporal query evaluation is based on the overlaying procedure that is introduced in the section 4.1. For a given user request U, the procedure first locates a set of roots from the root array of S^{PPF} such that the alive time interval of the root is overlapped with $[U.\tau_b, U.\tau_e]$. Then, for each located root r, the procedure traverses the subtree under this root r until it reaches the leaf level During this traversal, it compares the spatiotemporal extent of user request with that of each node in the search path. One would encounter three different cases:

- **enclosing:** If there exists any α such that the set of subjects evaluated by *ce* contains $U.s$, then return all the moving objects that are overlapped with $(N^{\square} \cap \alpha^{\square})$. In the case of the *locate* access mode, return the location information of those objects at time = $U.\tau_b$. $U.\tau_e$ is ignored if it is different from $U.\tau_b$ because, if we allow time interval, the trajectory information between the time interval $[U.\tau_b, U.\tau_e]$ is rather revealed to the user instead of the location information. If it is a *track* access mode, return the trajectory information of those objects. The trajectory of each object o in the result set is traced back by using the pointer $N.ptr$ where N is the leaf node that stores o. Whenever a node L is time split, ptr of newly created node is set to point back to the original node L. Thus, all the past location information of o can be reached by following the ptr created each time a node is split. This tracking is processed within the spatiotemporal region $U^{\square} \cap \alpha^{\square}$ where $\alpha^{\square}$ is the spatiotemporal region of all the authorizations with track privilege that are applicable to $U.s$.
- **overlapping:** If there exists any α such that the set of subjects evaluated by *ce* contains $U.s$, return the objects overlapping with $U^{\square}$ only. However, we still need to check authorizations overlaid for the descendents of the node N because authorizations overlaid for the descendents may include another spatiotemporal region that $\alpha^{\square}$ does not cover. In the case of the leaf node, return all the moving objects that are overlapped with $(N^{\square} \cap \alpha^{\square} \cap U^{\square})$. Again, if it is a *track* access mode, return the trajectory information of those objects. If it is a *locate* access mode return the location information.
- **disjoint:** Stop the evaluation process because no relevant authorizations can be found in the descendents of the node N to satisfy the request.

In SOR, we use $V.s$, $V.o$, and V^{point} to denote the subject, the requested auth-object and the spatiotemporal position of V, respectively. Only the alive root node among the root array is traversed because the access control request is evaluated only by the $V.s$'s location at t_c. The algorithm traverses the alive tree from the alive root node until it reaches the leaf level. During the traversal, it checks if the spatiotemporal extent of each node in the search path includes the V^{point}. If so, the procedure collects all the auth-objects contained in the set of ge such that $V.s$ is in the set of subjects evaluated by ce. If o is among this auth-objects set, stop traversing and return true, which means that the user s is allowed to get access to o. Otherwise, continue traversing. In case N is a leaf node, for each authorization N in α_S, include the auth-objects of α if $N^{\square} \cap \alpha^{\square}$ encloses V^{point} since we do not want to get the false positive result for the area $N^{\square}$ - $\alpha^{\square}$.

5 Related Work

Recently, there has been some endeavors to support effectively evaluating queries of *past, present* and *future* locations of moving objects. Patel et al. [7] index the positions that result from the dual data transformation. Lin et al. [8] propose the BB^x-index structure that inherits the ability to index present and future positions from the B^x-tree [9], and it extends this ability with support for also past

positions. However, BB^x-index does not correct trajectory segments and therefore, the index may include disconnected trajectories. Pelanis et al. [4] address this problem by applying partial persistence to the TPR-tree [6] and correcting the last-recorded prediction of moving object.

An index scheme for moving object data and user profiles has been proposed by Atluri et al. [10], but this does not consider authorizations. Beresford et al. [11], [12] have proposed techniques that let users benefit from location-based applications, while preserving their location privacy. Mobile users, in general, do not permit the information shared among different location based services. Primarily, the approach relies on hiding the true identity of a customer from the applications receiving the user's location, by frequently changing pseudonyms so that users avoid being identified by the locations they visit. A system for delivering permission-based location-aware mobile advertisements to mobile phones using Bluetooth positioning and Wireless WAP Push has been developed [13]. An index structure has been proposed to index authorizations ensuring that the customer profile information be disclosed to the merchants based on the choice of the customers [14]. However, this provides separate index structures for data and authorizations, and therefore is not a unified index.

Atluri and Guo have proposed a unified indexing scheme for moving objects and authorizations, called ^{S}TPR-tree. However, because ^{S}TPR-tree does not maintain historical information on moving objects, it does not support security policies based on tracking of mobile users. The focus of this paper is to provide persistence for unified indexing scheme for mobile objects and authorizations.

6 Conclusions

A number of services in the area of mobile commerce environment require maintaining mobile object's location information and efficiently serving access requests on *past, present* and *future* status of the moving objects. Proper access control policies must be enforced to address the security and privacy concerns in this environment. Recently, Atluri and Guo have proposed an unified index structure, ^{S}TPR-tree, to organize both the moving objects and authorizations specified over them. However, the ^{S}TPR-tree is not capable of answering queries based on the past information, and therefore cannot support security policies based on *tracking* of mobile users. In this paper, we have proposed an index structure, called the S^{PPF}-tree, which maintains past, present and future positions of the moving objects along with authorizations by employing the *partial persistent storage*, and therefore can support authorizations based on tracking of mobile objects. Currently, we are conducting a performance evaluation to demonstrate that our uniform indexing scheme indeed has significant impact on the response time.

In our proposal, an authorization is based on the spatial and temporal attributes of either subjects or objects. Thus, we are not capable of overlaying authorizations that cannot be represented with spatiotemporal region. Also, our overlaying strategy cannot accommodate authorizations whose subjects and ob-

jects are both moving at the same time. As a result, supporting such authorizations overlaying may require splitting the subject and object components. We will enhance our S^{PPF}-tree to address these issues. Support for negative authorizations require significant changes to the overlaying of authorizations as well as evaluating access requests. In this paper, we do not consider negative authorizations; we will extend our work to support negative authorizations.

References

1. Wolfson, O., Xu, B., Chamberlain, S., Jiang, L.: Moving objects databases: Issues and solutions. In Rafanelli, M., Jarke, M., eds.: 10th International Conference on Scientic and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998, IEEE Computer Society (1998) 111-122
2. Moreira, J., Ribeiro, C., Abdessalem, T.: Query operations for moving objects database systems. In: Proceedings of the eighth ACM international symposium on Advances in geographic information systems, ACM Press (2000) 108-114
3. Atluri, V., Guo, Q.: Unied index for mobile object data and authorizations. In: ESORICS. (2005) 80-97
4. Pelanis, M., Saltenis, S., Jensen, C.S.: Indexing the past, present and anticipated future positions of moving objects. a TIMECENTER Technical Report TR-78 (2004)
5. Atluri, V., Chun, S.A.: An authorization model for geospatial data. IEEE Trans. Dependable Sec. Comput. 1(4) (2004) 238-254
6. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: SIGMOD Conference. (2000) 331-342
7. Patel, J.M., Chen, Y., Chakka, V.P.: Stripes: an efficient index for predicted trajectories. In: Proceedings of the 2004 ACM SIGMOD International conference on Management of data, New York, NY, USA, ACM Press (2004) 635-646
8. Lin, D., Jensen, C.S., Ooi, B.C., Saltenis, S.: Efficient indexing of the historical, present, and future positions of moving objects. In: Mobile Data Management. (2005) 59-66
9. Jensen, C.S., Lin, D., Ooi, B.C.: Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In: VLDB. (2004) 768-779
10. Atluri, V., Adam, N.R., Youssef, M.: Towards a unied index scheme for mobile data and customer proles in a location-based service environment. In: Workshop on Next Generation Geospatial Information (NG2I'03). (2003)
11. Beresford, A., Stajano, F.: Mix zones: User privacy in location-aware services. In: PerCom Workshops. (2004) 127-131
12. Scott, D., Beresford, A., Mycroft, A.: Spatial security policies for mobile agents in a sentient computing environment. In: FASE. (2003) 102-117
13. Aalto, L., Gthlin, N., Korhonen, J., Ojala, T.: Bluetooth and wap push based location-aware mobile advertising system. In: MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services, New York, NY, USA, ACM Press (2004) 49-58
14. Youssef, M., Adam, N.R., Atluri, V., eds.: Preserving Mobile Customer Privacy: An Access Control System for Moving Objects and Customer Information. In 6th International Conference on Mobile Data Management. Lecture Notes in Computer Science, Springer (2005)