A Compressed Accessibility Map for XML

TING YU
North Carolina State University
and
DIVESH SRIVASTAVA
AT&T Labs–Research
and
LAKS V.S. LAKSHMANAN
University of British Columbia
and
H. V. JAGADISH
University of Michigan, Ann Arbor

XML is the undisputed standard for data representation and exchange. As companies transact business over the Internet, letting authorized customers directly access, and even modify, XML data offers many advantages in terms of cost, accuracy, and timeliness. Given the complex business relationships between companies, and the sensitive nature of information, access must be provided selectively, using sophisticated access control specifications. Using the specification directly to determine if a user has access to an XML data item can be extremely inefficient. The alternative of fully materializing, for each data item, the users authorized to access it can be space-inefficient. In this paper, we introduce a compressed accessibility map (CAM) as a space- and time-efficient solution to the access control problem for XML data. A CAM compactly identifies the XML data items to which a user has access, by exploiting structural locality of accessibility in tree-structured data. We present a CAM lookup algorithm for determining if a user has access to a data item that takes time proportional to the product of the depth of the item in the XML data and logarithm of the CAM size. We develop an algorithm for building an optimal size CAM that takes time linear in the size of the XML data set. While optimality cannot be preserved incrementally under data item updates, we provide an algorithm for incrementally maintaining near-optimality. Finally, we experimentally demonstrate the effectiveness of the CAM for multiple users on a variety of real and synthetic data sets.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*query processing*; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*

General Terms: Algorithms, Experimentation, Performance, Theory

Authors' email addresses: T. Yu (yu@csc.ncsu.edu); D. Srivastava (divesh@research.att.com); L.V.S. Lakshmanan (laks@cs.ubc.ca); H.V. Jagadish (jag@umich.edu).
T. Yu's research was largely done when he was at the University of Illinois, Urbana-Champaign, and was supported by DARPA via AFRL contract F33615-01-C-0336 and via Space and Naval Warfare Systems Center San Diego (SSCSD) grant N66001-01-18908. L.V.S. Lakshmanan's research was supported by grants from NSERC and NCE/IRIS. H.V. Jagadish's research was supported in part by NSF under grants DMI-0075447 and IIS-0208852.

Additional Key Words and Phrases: Access control, structural locality, XML

1. INTRODUCTION

The eXtensible Markup Language (XML) is widely regarded as the undisputed standard for data representation and exchange, due in large part to its simplicity and capability of representing rich data structures. As companies transact business over the Internet, letting authorized customers directly access, and even modify, operational data items in XML documents over the Web offers many advantages in terms of cost, accuracy, and timeliness. However, it raises the question of security. Given the sensitive nature of business information, access must be provided selectively. Furthermore, the nature of this selective access has more and more sophisticated requirements imposed on it, as we move from B2C (business-to-consumer) e-commerce to B2B (business-to-business) e-commerce. For instance, a large corporation may be willing to let a supplier view (i) inventories for parts that it supplies (but not for other parts), and (ii) production schedules for items manufactured using this supplier's parts, but only up to the next two weeks, etc. Where suppliers participate in the design process, as many suppliers to the big three American automobile companies do today, a complex collection of design data must be shared with appropriate caution. The sophistication of the associated access control policy required is appreciable.

While there are certainly security issues with regard to obtaining protection against various forms of attack on the Internet, *our interest in this paper is in efficiently enforcing a stated access control policy over XML documents.*

We expect that the policy will be specified in terms of a (potentially large) set of access control rules, which specify user access to data items on the basis of user properties and categories, and on the basis of the properties of the data items in question. These rules frequently interact with one another in complex ways, with some priority mechanism or conflict resolution mechanism used for unambiguously deciding for a user, an item, and an access type whether the user has access to it or not. As a consequence, using the access control specification *directly* to determine whether or not a user has access to a particular data item can be extremely inefficient. This inefficiency is compounded for complex queries, where it does not suffice to check only at the end of query evaluation whether the user is authorized to access each data item in the final query result; one may need to perform an accessibility check individually for each data item touched by the query during query evaluation.

One possible approach for efficiently determining whether or not a user has access to a particular data item is to build a fully materialized accessibility map, which maintains, for each data item, the set of users authorized to access it; this can, however, require a large amount of space.

In this paper, we propose the *compressed accessibility map* (CAM) as a means of recording exactly this information, but in a much smaller space, for XML documents, independent of how the XML documents are stored. Given a user and a data item, the CAM can be used to determine efficiently if a user has access to the

data item. Our key contributions are as follows:

— We propose the novel notion of a compressed accessibility map (CAM) as a space- and time-efficient solution to the access control enforcement problem for XML data. This compression is obtained by exploiting structural locality of accessibility, i.e., data items grouped together have similar accessibility properties, on a per-user basis. The total size of the CAM is typically a small fraction of the fully materialized accessibility map.

— We develop an algorithm to construct an optimal (minimum size) per-user CAM for a given accessibility map, which takes time linear in the database size (number of items in the database for which accessibility is individually identified).

— Using an optimal CAM, we present algorithms for efficiently determining if a user has access to a data item. This lookup takes time proportional to the product of the depth of the item in the XML data and a logarithm of the size of the CAM.

— We show that, while optimality of the CAM cannot be preserved incrementally under data item updates, near-optimality (optimal size +1) can be preserved. We provide an algorithm for doing so that runs in time proportional to the size of the update applied. This price need be paid only once per batch if updates are suitably batched.

— Finally, we experimentally demonstrate the effectiveness of the CAM for hundreds of users on a variety of real and synthetic data sets, validating our approach to the access control enforcement problem.

The problem setting and the basic idea of the CAM are described in Section 2. Optimal CAM construction is addressed in Sections 3, 4, and 6. CAM lookup is developed in Sections 5 and 6. Experimental results are reported in Section 7. Incremental maintenance is considered in Section 8. Related work is discussed in Section 9, just before summarizing our conclusions in Section 10.

2. THE PROBLEM SETTING

2.1 Accessibility Map

Conceptually, an access control policy induces a projection of the tree representation of an XML database (XML database tree, for short) for each user, consisting of the nodes the user is allowed to access according to the policy. For example, the portion of the XML document in Figure 1 that can be accessed by those in the public domain is given in Figure 2.[1] The policy itself may be specified in terms of access control rules with conflict resolution mechanisms, explicit access control lists, or any other means appropriate. The specification mechanism is orthogonal to the efficient enforcement techniques of this paper.

Moreover, there could be many different types of access allowed, e.g., read, modify, append, etc, and policies may also specify some relationships between these access types. For instance, a user given modify access to a node may automatically also be granted read access. As far as we are concerned, all such policies are also folded in to create a final decision (of "accessible" or "inaccessible") for each node,

[1]Figures 1 and 2 are based on examples in [Damiani et al. 2000b]. The semantics of the `access` attribute is explained in Section 2.2.

for each user, and for each access type. We refer to this as the *accessibility map* of the database tree, formalized below.

Definition 2.1. (Accessibility Map) Let H be the set of nodes in the XML database tree, U be a set of users, and A be a set of access types. The *accessibility map* is given by a function $M : H \times U \times A \rightarrow \{\text{accessible}, \text{inaccessible}\}$. □

A direct enforcement of the policy will require determination of the relevant portion of the accessibility map at run time. For systems of any size, there will typically be a number of potentially relevant policy specifications to consider, making such run-time determination quite often too inefficient. On the other hand, the fully materialized accessibility map (suitably indexed) supports rapid determination of accessibility, but can be very space-inefficient. In fact, for small units of access control, fully materializing the entire accessibility map may not be considered feasible at all due to the high space overhead involved.

Our goal is to identify a compact means for recording this marking, which yet affords quick lookup.

2.2 Hierarchically Structured Data and Structural Locality

A key foundation of our work is *structural locality of accessibility*, i.e., data items grouped together have similar (but not necessarily identical) accessibility properties, on a per-user basis. This is commonly observed in hierarchically-structured data used in supporting e-commerce applications, such as XML documents and file systems, where the structural organization typically results in a hierarchical grouping of closely related data items. Locality of accessibility exists not only horizontally, i.e., between entities that share the same parent, but also vertically, i.e., between parents and children. As a consequence, if an entity is accessible, then very likely (but not always) so are its ancestors.

XML has a hierarchical Document Object Model [Fernandez et al. 2002] within a document. Access control is often desired at a granularity much finer than an entire XML document, perhaps even down to the level of individual elements in some cases [Bertino et al. 1999; Damiani et al. 2000a]. The access control policy is typically specified declaratively, using access control rules and conflict resolution mechanisms. Many proposed access control models for XML documents allow users to propagate an access control policy in a data item to its descendents unless it is overridden by more specific access control policies. See, for example, the XML document in Figure 1, based on the example in [Damiani et al. 2000b], where sub-elements inherit the access policy of their parent element (specified by the value of the `access` attribute), but can also choose to override it. Note that the access level of the root element <`division`> is explicitly specified as "public" (meaning, accessible to all). This access level is inherited by its sub-elements <`about_div`> and <`res_activity`>, but overridden by its <`seminar`> sub-elements, which explicitly specify their access levels as "internal" (meaning, accessible to some specified subset of the users). Similarly, all sub-elements of the <`seminar`> elements inherit their access levels. Therefore, structural locality of accessibility is quite natural in XML documents.

In addition, multiple XML documents in a data store may themselves also be organized hierarchically, for instance, using a file system with a hierarchical directory

```
<division name = "Security" access = "public">
   <about_div>
      <member>
         <name> Bob </name>
         <position> Computer Scientist </position>
         <e-mail> bob@acme.com </e-mail>
      </member>
      <member>
         <name> Tom </name>
         <position> Software Engineering </position>
         <e-mail> tom@acme.com </e-mail>
      </member>
      <contact>
         Security Division - 180 Lane St - 81231 New Park
      </contact>
   </about_div>
   <res_activity>
      <description access = "internal"> The purpose of ...</description>
      <project access = "public" type = "system">
         <name access = "internal"> Access Control </name>
         <fund access = "internal">
            <sponsor> IT </sponsor>
            <amount> 10000 </amount>
         </fund>
         <report code ="R1-99" access = "internal">
            <title> A new access control model </title>
            <author> Sam </author>
            <author> Ron </author>
            <text> ......  </text>
         </report>
      </project>
      <project access = "public" type = "theory">
         <name> Cryptography </name>
         <report code ="R2-99" access = "public">
            <title> The study of encryption </title>
            <author> Steve </author>
            <text> ......  </text>
         </report>
      </project>
   </res_activity>
   <seminar access = "internal">
      <date> Tues., June 8 </date>
      <title> Safe statistics </title>
      <speaker> Jan </speaker>
   </seminar>
   <seminar access = "internal">
      <date> Thurs., July 15 </date>
      <title> UML </title>
      <speaker> Karen </speaker>
   </seminar>
</division>
```

Fig. 1. An example XML document with access control annotations

```
<division name = "Security">
   <about_div>
      <member>
         <name> Bob </name>
         <position> Computer Scientist </position>
         <e-mail> bob@acme.com </e-mail>
      </member>
      <member>
         <name> Tom </name>
         <position> Software Engineering </position>
         <e-mail> tom@acme.com </e-mail>
      </member>
      <contact>
         Security Division - 180 Lane St - 81231 New Park
      </contact>
   </about_div>
   <res_activity>
      <project type = "system">
      </project>
      <project type = "theory">
         <name> Cryptography </name>
         <report code ="R2-99">
            <title> The study of encryption </title>
            <author> Steve </author>
            <text> ......  </text>
         </report>
      </project>
   </res_activity>
</division>
```

Fig. 2. Publicly accessible view of the document in Figure 1

structure. Closely related files are placed in a single directory, or a set of sibling directories in popular operating systems, such as UNIX and Windows. Current approaches, such as creating user groups with common access rights, has the potential consequence of creating a very large number of groups to support the complexity of access control in e-commerce applications. If the number of groups is restricted, then one needs the ability to specify access rights for multiple groups on the same resource, leading to very large data structures.

We assume that we are given an XML database tree H, each of whose nodes is uniquely identified by a hierarchically structured identifier that allows direct determination of parent-child and ancestor-descendant relationships between tree nodes. In this paper, we use the identifier that is the concatenation, in root to leaf order, of node positions (in the prefix traversal of the tree) in the path to it from the root of the tree.[2] This is akin to fully qualified file names in UNIX, or the Dewey Decimal Classification used by librarians. The nodes of this tree reflect the

[2]If the database is a forest, our techniques can easily be applied by introducing a dummy root node and considering the forest to be a single tree.

finest granularity of accessibility.

2.3 Compressed Accessibility Map

The key to obtaining a space-efficient representation for the accessibility map is to exploit the structural locality of accessibility in the XML database tree, and maintain a separate *compressed accessibility map* (CAM), for each user and access type. In the bulk of this paper, we deal with per-user, per access type CAMs, and experimentally demonstrate the feasibility of this approach for multiple users and access types.

Restricted to a single user and access type, the accessibility map is simply a *marking* that maps the nodes of the XML database tree to the set {accessible, inaccessible}. We call a database tree H together with a marking M as a *marked tree*. The basic idea of the CAM is that, instead of explicitly keeping a list of all the accessible (or inaccessible) nodes, we keep only some "crucial" nodes and place some additional information on them so that we can efficiently check whether an arbitrary node can be accessed or not by simply looking at relevant "crucial" nodes. The compression achieved by CAM relies on two observations.

First, in a "region" of the XML database tree, uniform accessibility of descendants of a node can be represented at the node itself. For instance, if each node in a subtree of the marked tree is accessible, substantial compression can be achieved if the CAM maintains the root node of the subtree with a $(d+, s+)$ label, indicating that the node is accessible ($s+$), and so are its descendants ($d+$).[3] This observation can be generalized to employ a simple node labeling mechanism, involving $d+, d-, s+$ and $s-$. The semantics of the labels is as follows. If node x carries an $s+$ (resp., $s-$), then x is accessible (resp., inaccessible). If node x carries a $d+$ (resp., $d-$) and y is a descendant of x, then y is accessible (resp., inaccessible), unless this is overridden by the label of a closer ancestor of y (or by the label at y itself).

Second, it is frequently the case that, at least within a "local region" of the XML database tree, there is a hierarchical aspect to accessibility: if a node is accessible, then so are its ancestors. We call this the *ancestor accessibility* property. This observation can be utilized in the CAM as follows: if a node x carries an $s+$, then so must all its ancestors, within the local "region".

In many access control models for XML documents, a user is required to have rules that, explicitly or implicitly, propagate an element's accessibility to its descendents. Therefore, the ancestor accessibility property holds across the entire document. This is the case for the example XML document of Figure 1.

When the ancestor accessibility property does not hold over the entire marked database tree, it is easy to partition the tree into regions that satisfy the ancestor accessibility property. We call each resulting database subtree a *unit region*. An example XML document that does not satisfy the ancestor accessibility property is shown in Figure 3. Note that the access level of the root element <`contracts`> is "private" (meaning, accessible to a specified subset of users), while one of its descendents <`news`> has access level "public". Therefore, the element <`news`> is

[3]Note that the CAM only serves to indicate whether or not a named node is accessible, not whether or not a named node exists in the database tree.

```
<contracts access = "private">
   <contract number = "200212">
      <value> 275 </value>
      <name> Acme Co </name>
      <legaldoc href = "acme.lgl"/>
      <news href = "acme.html" access = "public"/>
   </contract>
   <contract number = "200213">
   ...
</contracts>
```

Fig. 3. XML document without ancestor accessibility

visible to all users, but its ancestors are not.

The algorithm to partition a given marked database tree into unit regions is very simple: Perform a traversal of the database tree, and mark each node u that is accessible but the parent node of u is inaccessible. Each such node is called a *marker* node, since it demarcates the boundary of a unit region. The subtree rooted at a marker node (or at the root node of the entire database tree) is a unit region, excluding any descendant marker nodes and their associated subtrees. In the example of Figure 3, each <news> element is a marker node, and in a unit region by itself; the rest of the document is a single unit region. In general, we expect the number of unit regions in a database tree not to be too large. Consequently, it makes sense to exploit the ancestor accessibility property to obtain compression within each unit region.

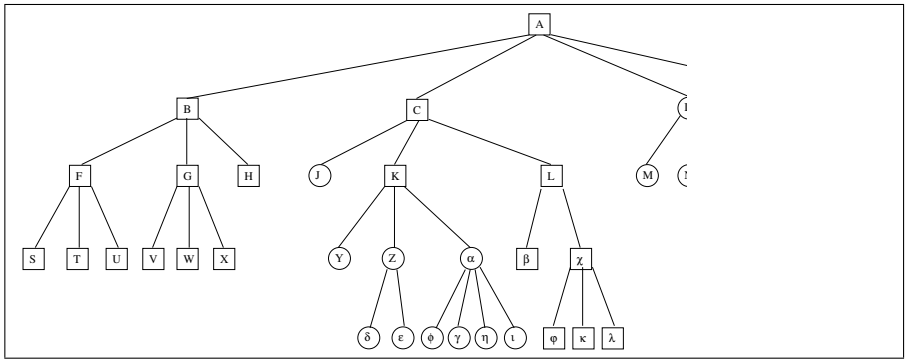
For expositional reasons, we first develop the compressed accessibility map for a single unit region, in the next several sections, and then, in Section 6, deal with database trees comprised of multiple unit regions.

3. UNIT REGION CAM

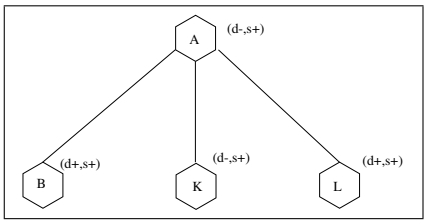
Not all nodes need be labeled explicitly, since accessibility of many nodes can be determined by the d label of its closest labeled ancestor. In addition, some other nodes do not require explicit labels on account of the ancestor accessibility property, as we shall see below. Moreover, in view of the semantics of these labels and the ancestor accessibility property, it will follow that the label $(d+, s-)$ can always be replaced by $(d-, s-)$. (If a node is not accessible, none of its descendants (within the unit region) can be accessible.)

Example 3.1. Consider again the XML document (with the access control policy) of Figure 1. The marked database tree, which is an abstract representation of this XML document, is shown in Figure 4(a), where square nodes are accessible and round nodes are not. The CAM of the tree is shown in Figure 4(b). Notice how much smaller this is compared to the XML view in Figure 2.

Nodes Y, Z, α and all their descendents can be inferred to be inaccessible on account of the $d-$ label at their nearest labeled ancestor, K. Node K itself can be inferred to be accessible because of its own $s+$ label. Node C would have been considered inaccessible on account of the $d-$ label at its nearest labeled ancestor, A. Nonetheless, the $s+$ label at C's child, K, causes C to be inferred to be accessible



(a) A marked database tree; square nodes are accessible, round nodes are not



(b) An optimal CAM

Fig. 4. Compressing a marked database tree corresponding to the XML document in Figure 1

as well. □

A *labeling* K of a database tree H is a partial function that assigns labels of the form $(d+, s+)$, $(d-, s+)$, or $(d-, s-)$ to (some) nodes in H. We call the database tree along with its labeling, a *labeled database tree*, and denote it $\mathcal{T}$. Labeling K is complete whenever it is total, that is, each node is labeled. Given such a labeling, we can determine which nodes are accessible and which are not, based on the $s+$ or $s-$ label at the node, respectively. A (complete) labeling K is said to *respect* a marking M of a database tree H, if for each node in H, it is determined to be accessible by K exactly when it is accessible according to its marking in M. In the sequel, by a (complete) labeling for a marked tree, we mean one that respects the marking.

It is worth noting that there may be multiple complete labelings that respect any given marking of a database tree. For example, a marked database tree that has all its nodes accessible is respected both by the labeling that assigns label $(d+, s+)$ to each node, as well as by the labeling that assigns label $(d-, s+)$ to each node.

As mentioned previously, due to the semantics of a d label and the ancestor accessibility property in a unit region, it is possible that a node's label is redundant, in the sense that, by removing its label, the resulting labeling still respects the

marking of a database tree. The notion of induced labels, defined next, plays a role in determining which node labels in a labeled database tree are redundant.

Definition 3.2. (Induced Label) The *induced label* on a node e in a labeled database tree, written $L(e)$, is its label if one exists. Else, for unlabeled node e,

—If the nearest labeled ancestor of e has a label $(d+, s+)$, then $L(e) = (d+, s+)$.
—If the nearest labeled ancestor of e has a label $(d-, s-)$, then $L(e) = (d-, s-)$.
—If the nearest labeled ancestor of e has a label $(d-, s+)$,
  —If e has a descendant labeled with either $(d-, s+)$ or $(d+, s+)$, then $L(e) = (d-, s+)$
  —Else $L(e) = (d-, s-)$.
—If e has no labeled ancestor, then $L(e)$ is undefined. □

Note that in examining the labels of ancestors and descendants, only original labels should be considered, not induced labels. It is easy to see that the induced label on a node is uniquely defined, and does not depend on the order in which nodes are considered.

For example, given the labeling shown in Figure 4(b), the induced labels at (some of the) nodes in Figure 4(a) are as follows: B, F and G get $(d+, s+)$, A, K and C get $(d-, s+)$, and J, D and E get $(d-, s-)$.

Definition 3.3. (Accessible Node) A node e in a labeled database tree $\mathcal{T}$ is *accessible* if (i) it has an induced label of $(d+, s+)$ or $(d-, s+)$, or (ii) its induced label is undefined, and it has an accessible child node. Node e in $\mathcal{T}$ is inaccessible otherwise. □

For example, it is easily verified that, given the labeling shown in Figure 4(b), the accessible nodes of Figure 4(a) are precisely the square nodes.

Definition 3.4. (CAM) Two labelings over a database tree H are said to be *equivalent* if for each node e in H, e is accessible according to the first labeling iff e is accessible according to the second.

We say a labeling $\mathcal{L}$ is a *restriction* to another labeling $\mathcal{L}'$ if $\mathcal{L}$ is a subset of $\mathcal{L}'$.

A *compressed accessibility map* (CAM) associated with a complete labeling is a restriction to it that is equivalent.

A CAM for a marked database tree is a CAM associated with any complete labeling that respects the given marking. □

In short, we take the complete labeling, and leave some labels off to obtain a CAM. Note that the induced labels from the CAM may not recover the original labeling exactly — what is important is that the *accessibility* is determined correctly for each node.

Given a CAM (i.e., labeling) K of a marked database tree H, we can represent the CAM succinctly as a *reduced graph*, defined as follows: (i) it contains only nodes from H that are labeled in K, and (ii) for two labeled nodes x, y from H, there is an edge from x to y in the reduced graph iff x is the closest labeled ancestor of y in H, i.e., x is an ancestor of y that is labeled in K, and there is no labeled node z such that z is a (proper) ancestor of y but a (proper) descendant of x in H. In

general, the reduced graph of H is a forest instead of a tree. In that case, we add a dummy root to render it a tree. In the sequel, we identify the CAM with its reduced tree representation.

For example, consider the marked database H shown in Figure 4(a). Then the reduced tree of the labeling (CAM) K shown on the nodes of Figure 4(a) is as shown in Figure 4(b).

The next question is which labels can we leave out, while still determining unambiguously the accessibility of each node. We do this characterization using the notion of redundant labels below.

Definition 3.5. (Subsumed Label) A label at node e in a CAM $\mathcal{I}$ is said to be *subsumed* if it is identical to the induced label at e in a CAM $\mathcal{I}'$ obtained from $\mathcal{I}$ by rendering e unlabeled. □

Definition 3.6. (Upward Redundant Label) A label at node e in a CAM $\mathcal{I}$ is said to be *upward redundant* if

—e has an accessible (proper) descendant, and

—for every child c of e, either c is labeled in $\mathcal{I}$ or c is upward redundant. □

The intuition here is that the accessible descendant induces an $s+$ at e, and since all children are either labeled or upward redundant, the d-label at e is immaterial. So we do not need to record a label at e. Note that the d-label that e has, and the exact labels its children have (if at all), are immaterial as far as upward redundancy is concerned.

Definition 3.7. (Redundant Label) A label at node e in a CAM $\mathcal{I}$ is said to be *redundant* if it is either subsumed or upward redundant. □

Obviously, in any optimal CAM, no redundant labels are retained. How to construct one efficiently is the subject of the next section.

4. OPTIMAL UNIT REGION CAM

The reduced tree representation of the CAM leads to the following natural definition:

Definition 4.1. (CAM Size) The *size* of a CAM is the number of labeled nodes in the CAM. □

Since the number of users with differing accessibility to the XML data can be large, it is important to minimize carefully the amount of storage required for the CAM of each user. Storage is determined by the number of labeled nodes. A given accessible projection of a database H can be represented by several equivalent labelings (CAMs). Our task is to determine a CAM of the smallest size. Conceptually, we can start with any complete labeling and delete all redundant labels. However, this naive strategy is not guaranteed to produce an optimal size CAM, owing to the following two complications.

First, given any complete labeling (that respects the database marking), it is not clear what is the order in which we should delete redundant labels: deleting one could render another no longer redundant. Consider, for example, a marked database tree that is a chain of two accessible nodes, and the labeling that labels

each node $(d+, s+)$. The root's label is upward redundant, while the child's label is subsumed. Deleting either of these labels renders the other non-redundant. Indeed, in general, there are exponentially many orders in which to delete redundant labels.

Second, there are equivalent complete labelings, such that the minimal CAM afforded by one need not be the same size as the minimal CAM afforded by another. Consider, for example, the marked database tree with every node accessible, and the two complete labelings: $\mathcal{I}_1$, which labels each node by $(d+, s+)$, and $\mathcal{I}_2$, which labels each node by $(d-, s+)$. It is easy to see that these labelings are equivalent. The minimal CAM for $\mathcal{I}_1$ only labels the root node of the database tree with $(d+, s+)$. The minimal CAM for $\mathcal{I}_2$ needs to label the root and each leaf node of the database tree with $(d-, s+)$. Clearly, these (minimal CAMs) are not of the same size.

Thus, finding an optimal CAM for a given database tree is a non-trivial problem. We solve it by establishing some key properties of label redundancy, and use them to devise an efficient algorithm for constructing an optimal CAM.

4.1 Order of Redundancy Removal

LEMMA 4.2. *(Subsumption Order Invariance) Given a CAM $\mathcal{I}$ in which the labels at nodes e and f are subsumed, it must be the case that the label at f is subsumed in the CAM $\mathcal{I}'$ obtained from $\mathcal{I}$ by making e unlabeled.*

PROOF. By definition of subsumption, the unlabeling of e can affect the subsumption of the label at f only if e is the nearest labeled ancestor of f in $\mathcal{I}$. Suppose this is the case.

Let g be the nearest labeled ancestor of e in $\mathcal{I}$, and hence in $\mathcal{I}'$. Then g is also the nearest labeled ancestor of f in $\mathcal{I}'$. Considering the different cases in the definition of induced label, it is easy to see that the induced label at f does not change on account of the unlabeling of e. In all cases, but one, this is because the labeling at g is the same as the labeling at e. The only exception is when e has an induced label of $(d-, s-)$ with g having a label of $(d-, s+)$. In this case, f has an induced label of $(d-, s-)$. Moreover, f has no accessible descendants, since e does not. Now, with e unlabeled, we can still induce the $(d-, s-)$ label on f, from g, even though it has a label of $(d-, s+)$. □

It is also easy to see that the order in which different upward redundant labels are removed is immaterial, since the definition of upward redundancy simply speaks of nodes with upward redundant labels, without consideration to whether these labels are present or have been removed.

Definition 4.3. (Ancestor-free) A node e in a CAM $\mathcal{I}$ is said to be *ancestor-free* if no proper ancestor of e is labeled in $\mathcal{I}$. □

Definition 4.4. (CAM Building Rules) Let K be any labeling (complete or partial) of a database tree H. Given the two sources of redundancy: subsumption and upward redundancy, every CAM for K can be obtained by an application of the following rules in an arbitrary order, until no longer possible.

Rule 1:. Delete the label of node e from K whenever it is upward redundant.

Rule 2:. Delete the label of node e from K whenever it is subsumed.

It is convenient to use a regular expression notation that indicates the way a CAM was obtained. We denote a general CAM obtained as above, as a $(1+2)^*$-CAM for K, meaning it was obtained by a repeated applications of rules 1 and 2 in an arbitrary order. We also consider the following restricted version of rule 1:

Rule 1′:. Delete the label of node e from K whenever it is upward redundant and e is ancestor-free in K. □

It is obvious that each of these rules preserves equivalence and hence transforms a CAM into another (smaller) CAM. We have the following result:

THEOREM 4.5. *(Subsumption First) Given a CAM $\mathcal{I}$, let $\mathcal{I}'$ be a restriction to $\mathcal{I}$, obtained by deleting subsumed labels and upward redundant labels in a specified order. The smallest size $\mathcal{I}'$ is obtained by deleting all subsumed labels first, and then deleting ancestor-free upward redundant labels depth-first from the root in a pre-order tree traversal. More precisely, for every $(1+2)^*$-CAM for $\mathcal{I}$, there is a $(2^*1'^*)$-CAM which is equivalent and no larger.*

PROOF. Deleting an upward redundant ancestor-free label in $\mathcal{I}$ cannot render any other label in $\mathcal{I}$ subsumed when it previously was not.

Consider the relative ordering of deletion of a label at e that can be subsumed and a label at a node f that is ancestor-free and upward-redundant. e cannot be an ancestor of f. e and f may be unrelated by an ancestor-descendant relationship, in which case the removal of the label at one does not affect the other so the relative ordering is immaterial. The interesting case is when f is an ancestor of e.

Case 1: f is the nearest labeled ancestor of e.

Deleting label at e (via subsumption) could render f no longer upward redundant, but does not negatively impact upward redundancy of any other nodes. This is because the only labeled ancestor of e is f and thus unlabeling of e cannot have any influence on the upward redundancy of any other node. Deleting label at f renders e no longer subsumed. In addition, any possible subsumption of labels at siblings at e is also now prevented. Consequently, the cost of deleting f first is greater than equal to the cost of deleting e first.

Case 2: f is not the nearest labeled ancestor of e.

In this case, deleting label at f does not affect subsumption at e, and deleting label at e does not affect upward redundancy of f. So once again the order is immaterial.

Putting this together, we see that in all cases, if it is material, it is preferable to remove subsumed labels first and then upward redundant ancestor-free labels. Also, it is obvious for the latter class, that one must proceed from the root down (otherwise ancestor-freeness will not hold). □

4.2 The Optimal CAM Construction Algorithm

Definition 4.6. (Terminal Node) A node e in a CAM is called a *terminal* node if e is accessible but none of its descendants is. When a terminal node is not a leaf, we call it an *internal terminal* node. □

The following lemma shows that we can safely label all internal terminal nodes $(d-, s+)$, without compromising optimality.

LEMMA 4.7. *(Terminal Label) There exists an optimal labeling with every internal terminal node labeled* $(d-, s+)$.

PROOF. Let $\mathcal{I}$ be any CAM. We will show there is an equivalent CAM with size no more than that of $\mathcal{I}$, from which the lemma will follow.

Consider an internal terminal node e. e has at least one descendant node, but every one of them is not accessible, so if any of these descendants is labeled in $\mathcal{I}$, the s-label must be $s-$. So using $\mathcal{I}$, we can determine e is accessible either because e is labeled $s+$, or because its nearest ancestor, say f, is labeled $d+$.

Suppose the former is true, then e could either be labeled $(d+, s+)$ or $(d-, s+)$. Since every descendant of e is inaccessible, it is easy to see that a choice of $(d+, s+)$ is no better than $(d-, s+)$ (and possibly worse), so changing e's label to $(d-, s+)$ yields a CAM $\mathcal{I}'$ with the same or smaller size and equivalent to $\mathcal{I}$.

Suppose the latter is true, and e is unlabeled in $\mathcal{I}$. Then every child node g of e has a label $(d-, s-)$ in $\mathcal{I}$. (Otherwise, g would also be deduced accessible on account of the label at f. Also, recall that $(d+, s-)$ is not meaningful.) Now create a new CAM from $\mathcal{I}$, by placing a label $(d-, s+)$ on node e and removing labels from all its children. Since e is non-leaf, it has at least one child. Therefore, the new CAM at worst does not increase the number of labeled nodes (it may have decreased this number). So, $\mathcal{I}'$ so obtained is no larger than and is equivalent to $\mathcal{I}$. □

Definition 4.8. (Positive and Negative Nodes) A node e is *positive* (resp., *negative*) provided

—e is an accessible (resp., inaccessible) leaf, *or*

—e is an internal non-terminal node with more positive (resp., negative) than negative (resp., positive) children. □

According to Lemma 4.7, terminal nodes can always be labeled with $(d-, s+)$, without compromising optimality. Therefore, we do not need to consider terminal nodes when defining positive and negative nodes. In some sense, a terminal node can be considered as a neutral node.

A declarative description of the optimal CAM construction algorithm is in Figure 5. It follows that a node's labeling will be deferred iff it is an internal non-terminal node which is neither positive nor negative. Note that the algorithm first builds a specific complete labeling (out of the exponentially many possible), for which the minimal equivalent labeling is a *global optimum*.

Since counts of positive and negative children are required to label a node in Stage 1, a bottom-up (or a post-order tree traversal) procedure is suggested. A single traversal suffices for Stage 1, except possibly for the deferred labels. But each deferred label node is visited exactly one additional time. The rules in Stage 2 are procedurally best applied top-down. Once again, saturation with each rule requires consideration of each node in the database tree at most once. Adding this up, the total time required by the above optimal CAM construction algorithm is proportional to the number of nodes in the database tree. We illustrate the algorithm in the next section.

Algorithm Opt-UnitRegion-CAM //given a marked database tree
// Stage 1:
label each accessible leaf $(d+, s+)$;
label each inaccessible node $(d-, s-)$;
label each terminal internal node $(d-, s+)$;
for each non-terminal internal node
label it $(d+, s+)$ if it has more positive than negative children,
label it $(d-, s+)$ if it has more negative than positive children,
and defer labeling otherwise;
whenever a node gets a label, propagate it to deferred descendants;
when the root has an equal number of positive and negative children,
give it an arbitrary label: $(d+, s+)$ or $(d-, s+)$;

// Stage 2:
apply rule 2 (delete subsumed labels) to saturation;
apply rule $1'$ (delete ancestor-free upward redundant labels) to saturation;

Fig. 5. Constructing an optimal CAM

4.3 An Illustrative Example of Optimal Labeling Algorithm

Consider the database tree in Figure 4(a), which shows a marking in the form of square (for accessible) nodes and circle (for inaccessible) nodes, for a given user and access type.

Stage 1 of the algorithm produces the following labeling:

—All the nodes in the subtree rooted at B are with label $(d+, s+)$.
—J$(d-, s-)$.
—All the descendents of K are with label $(d-, s-)$.
—K$(d-, s+)$ (internal terminal).
—All the nodes in the subtree rooted at L are with label $(d+, s+)$.
—C(deferred, since it has one positive child L, one negative child J, and child K is an internal terminal node).
—All the nodes in the subtree rooted at D are with label $(d-, s-)$.
—All the nodes in the subtree rooted at E are with label $(d-, s-)$.
—A$(d-, s+)$.

This leads to the deferred label at C being eventually set to $(d-, s+)$, once the label of the root A is decided. In Stage 2, all labels are found redundant, and removed, except those at A,B,K,L.

4.4 Optimality

We have the following observations.

(1) For each internal node e, the label assigned by Stage 1 of our algorithm maximizes the number of subsumed children. More precisely, the number of subsumed children is no fewer than in any equivalent labeling.

(2) When a node has an equal number of positive and negative children, Stage 1 gives it a label which makes it subsumed, unless the node is the root.

The couple of observations above are easily proved, and can be used to establish the following optimality results.

LEMMA 4.9. *(Good Labelings) Let K be any complete labeling of a unit region database tree T. Let $\mathcal{I}$ be any CAM associated with K. Then there is an equivalent labeling K' produced by Stage 1 of our algorithm such that there is a CAM $\mathcal{I}'$ associated with K' which is no larger.*

PROOF. Pick any deepest node where the label Kdiffers from what K' would have assigned. (By deepest node, we mean that no descendant of this node has label Kdifferent from K'). We will show by induction that changing the label of this node according to K' would: (a) give an equivalent labeling and (b) allow a CAM for that labeling whose size is no more than that of $\mathcal{I}$.

If there is an accessible leaf e labeled $(d-, s+)$ in K, change it to (d+,s+). This will not increase the cost: if this label was subsumed, then the subsuming ancestor label can only be (d+,s+).[4] In this case, changing it to (d+,s+) leaves it subsumed.

Let e be an internal node. If it is inaccessible, its label can only be $(d-, s-)$ in all labelings. I.e. there is no choice there.

If e is accessible, and if it is terminal, a label of $(d+, s+)$ there would require its inaccessible children to be labeled [$(d-, s-)$, e.g.]. If further, e's label was redundant, it can only be because it is subsumed. In this case, if e's label is removed, the labels of e's children cannot be removed. Trading this for making e's label $(d-, s+)$ (and hence non-redundant) and its children's labels subsumed (by e) would give an equivalent labeling and will not increase the cost.

Suppose e is non-terminal (and accessible). Suppose e has unequal number of positive and negative children, say more positive. (The other case is symmetric). If e has the label $(d-, s+)$, then in $\mathcal{I}$, one of the following holds. (If e is labeled $(d+, s+)$, we have no issue, since it coincides with K').

Case 1: e's negative children are unlabeled in $\mathcal{I}$.

In addition, e may also be unlabeled. Changing e's label to $(d+, s+)$ would allow e's positive children to be unlabeled in $\mathcal{I}'$, by applying the subsumption rule. The cost of $\mathcal{I}'$ is then less than equal that of $\mathcal{I}$.

Case 2: e's positive children are unlabeled in $\mathcal{I}$.

Since the positive children are not subsumed, and assuming e is a deepest node at which K deviates from a labeling produced by Stage 1, the only possibility is that e's label was erased in $\mathcal{I}$ using upward redundancy. Subsequently e's positive children are subsumed by the label of some ancestor of e. In this case, changing e's label to $(d+, s+)$ would allow e's positive children to be unlabeled in $\mathcal{I}'$. The ancestor of e will in this case subsume e's label in K', allowing $\mathcal{I}'$ to have a cost no higher than that of $\mathcal{I}$.

Suppose e has an equal number of positive and negative children. Suppose e's label, say $(d+, s+)$, is not subsumed in K. This cannot be an advantage over a labeling K' (with e's label changed to $(d-, s+)$), where the same number of e's children are subsumed as in K, and where in addition, e's label is also subsumed.

[4]Under a modified notion of subsumption which would allow this label to be removed from K.

The lemma follows by a simple induction. □

LEMMA 4.10. *(Choice of complete labeling) Let L be any complete labeling. Then one of the following holds:*

—*There is a* $(2^*1'^*)$*-CAM of L produced by Stage 2 of our algorithm which is no larger than any CAM of L,* or

—*There is a complete labeling L' such that:*
  —*L' is equivalent to L.*
  —*There is a* $(2^*1'^*)$*-CAM of L' produced by Stage 2 of our algorithm which is no larger than any CAM of L.*

PROOF. If the CAM, say T, for L was obtained using only rules 1' and 2, then by Theorem 4.5, it follows that there is a $(2^*1'^*)$-CAM for L with no more size than T. So assume T uses rule 1 at least once. Let e be a node with an upward redundant label in L with a labeled ancestor f, such that rule 1 was applied to remove e's label in T. Assume wlog that f is the closest labeled ancestor of e. Call such a node e, a bad node and suppose e is a deepest bad node, i.e. e satisfies this property but none of its descendants does. Let S be the CAM obtained using Stage 2 of the algorithm, i.e. S is a $(2^*1'^*)$-CAM.

Case 1: $label(e) = (d+, s+)$.

If the label of f in L is $(d+, s+)$ too, then e's label could be erased via rule 2. If this holds for every node whose label was erased using rule 1, then S will erase all such labels too, but via rule 2. So, assume f's label is $(d-, s+)$ in L. (It cannot be $(d-, s-)$.) If e has more children labeled $(d+, s+)$ than those that are either labeled $(d-, s-)$ or are labeled $(d-, s+)$ and are non-terminal, then erasing e's label via upward redundancy will cause the non-terminal children with labels $(d-, s+)$ and those with label $(d-, s-)$ to be subsumed (by f), but not the children labeled $(d+, s+)$. Since CAM T was obtained by applying this step, in the subtree rooted at e, children of e with label $(d+, s+)$ will remain labeled in T, while e and e's other children will not. let S be the CAM obtained using Stage 2 of our algorithm. in the CAM S, nodes e and children of e which are either non-terminal and labeled $(d-, s+)$ or are labeled $(d-, s-)$, will remain labeled and none of the other children will. as far as descendants of e's children are concerned, inductively, no CAM can be smaller than S, in the subtrees rooted at each of these descendants. so T cant be smaller than S within that subtree, either. putting all of this together, within the subtree rooted at e, the CAM S produced by Stage 2 is no larger than CAM T obtained as above.

Suppose e has fewer children labeled $(d+, s+)$ than those that are either labeled $(d-, s-)$ or are labeled $(d-, s+)$ and are non-terminal. Consider the labeling L' obtained by changing e's label from $(d+, s+)$ to $(d-, s+)$. Clearly, this labeling is equivalent to L. On this labeling, the CAM S obtained using Stage 2 of the algorithm will have no more labels than T, in the subtree rooted at e. The argument is identical to the above.

If e has the same number of children of either kind, then consider the labeling $\mathcal{L}'$ which agrees with L everywhere except e has label $(d-, s+)$. clearly, L' is equivalent to L. since CAM T for L was obtained by removing e's label via upward redundancy and then using subsumption (by f) on e's children, T would retain the labels for

```
Algorithm LookUp-UnitRegion-CAM
  // given a CAM trie C, and a query node e in database H
  let f1 in C denote the nearest labeled ancestor-or-self of e (in H);
  let f2 in C denote the nearest labeled descendant (if any) of e (in H);
  if (f1 = e) return (e's accessibility as determined by the s* label of f1);
  else if (label of f1 is (d−, s−)) return (e is inaccessible);
  else if (label of f1 is (d+, s+)) return (e is accessible);
  else // either f1 does not exist or label of f1 is (d−, s+)
    if (there is no f2) return (e is inaccessible);
    else return (e is accessible); // by ancestor accessibility
```

Fig. 6. Looking up accessibility of node e using a unit region CAM trie

children of e labeled $(d+, s+)$ in the subtree rooted at e, but not those of e or its other children. (Some of the descendants of e's children may also retain their labels, as above.) let S be the CAM for L' obtained using Stage 2 of our algorithm. indeed, S would retain labels for exactly e's children labeled $(d+, s+)$ and but not those of e or its other children. the reason is e's label would be erased via subsumption (by f's label). (again, S may retain label for some of the descendants of these children.) in addition, inductively, T cant be smaller than S in the subtrees rooted at any descendant of a child of e. thus, within the subtree rooted at e, S is no larger than T.

Case 2: label(e) = $(d-, s+)$.

The argument for this case is symmetric to that for case 1.

Now, suppose L is an arbitrary complete labeling. apply the above proof construction to the bad nodes in L in a bottom-up fashion starting with deepest such nodes. at each step, one of the two cases above applies. inductively, we either get a modified, but equivalent, labeling for which Stage 2 produces a CAM no larger than T (which is an arbitrary, but fixed, CAM of L), or we get a CAM of L produced by Stage 2 which is no larger than T. When all bad nodes have been processed, one of the conclusions stated in the claim clearly holds. □

THEOREM 4.11. *(Optimality) Let H be any marked unit region database tree. Then the CAM obtained using our algorithm is of the smallest size among all CAMs associated with any complete labeling of T that respects its marking.*

PROOF. Let K be any complete labeling of T that respects its marking. By Lemma 4.9, the labeling K' produced by Stage 1 is equivalent to it and admits a CAM which is no larger than any CAM of K. From the construction involved in Lemma 4.10, it should be obvious that on a labeling K' produced by Stage 1, there exists a $(2^*1'^*)$-CAM which is no larger than any CAM of K'. (That is, on labelings coming from Stage 1, the construction in Lemma 4.10 proof will not look for a different labeling.) Together, these two imply the theorem. □

5. UNIT REGION CAM LOOKUP

Recall that the (hierarchically structured) identifier for any node in the CAM, just as for the database tree, is a prefix of the identifiers for each of its descendants. As

such, we can store the CAM as a trie, keeping for each node its string extension relative to its parent along with its label, and introducing additional nodes to "factor out" any parent-relative string extensions common between siblings. Where the relative string extensions are long, string B-tree techniques [Ferragina and Grossi 1999; Jagadish et al. 2000] can be used to condense these. Using these well-known data representations, it is possible to store a CAM in space that is proportional to the number of labeled nodes, independent of the database size.

Having the (trie representation of the) optimal CAM for a complete labeling K and given a node e in the database tree H, checking whether e is accessible can be done quite efficiently, using Algorithm LookUp-UnitRegion-CAM in Figure 6.

Intuitively, if node e is labeled in the CAM, we look up its label, and we are done. Otherwise, its accessibility can be determined by its nearest labeled ancestor and a nearest labeled descendant (if any). The latter is useful only when e's nearest labeled ancestor is labeled with $(d-, s+)$. In this case, e may be accessible because of the ancestor accessibility property in a unit region. If e has a labeled descendant f_2, then no matter what f_2's label is, e must be accessible. A detailed discussion is given in the proof of Theorem 5.1.

Finding the nearest labeled ancestor and descendants is particularly easy using the trie representation of the CAM. Given the (query) identifier string for the data item in question, traverse the trie beginning from the root. Once we get to a labeled node that has no labeled child that is a prefix of the query string, we have found the desired nearest labeled ancestor of the query node. (When we say "labeled child" of node u in the trie, we mean the nearest labeled descendant of u in the trie, ignoring any unlabeled nodes introduced on account of shared prefixes.) We simply read off the label. This node has a child corresponding to a suffix extension of the query string iff the query node has a labeled descendant. Thus, a simple trie lookup of the query node identifier is all it takes. This strategy takes time proportional to the product of the length of the query string and the logarithm of the size of the CAM.

The correctness and efficiency of this algorithm are established by the following theorem.

THEOREM 5.1. *(CAM Lookup) Given a CAM corresponding to a unit region database tree, Algorithm LookUp-UnitRegion-CAM correctly determines whether a specified node is accessible. Further, it does so in time proportional to the product of the depth of the node in the XML tree and logarithm of the CAM size.*

PROOF. By definition of the labeling, a node e in a labeled database tree $\mathcal{T}$ is accessible if and only if it has an induced label of $(d+, s+)$ or $(d-, s+)$, or if its induced label is undefined and it has an accessible child node.

Begin by examining the nearest labeled ancestor of e (possibly itself). If e is itself labeled, or if its nearest labeled (proper) ancestor is labeled $(d+, s+)$ or $(d-, s-)$, we know the induced label at e, and we are done. The induced label is $(d-, s-)$ if this is the actual label on e or on its nearest labeled ancestor, in which case we know that e is inaccessible. In all of the remaining cases above, e is accessible.

If e is itself unlabeled, and its nearest labeled ancestor has a label of $(d-, s+)$, then the induced label at e depends on whether it has an accessible descendent. That is, e is accessible if and only if it has an accessible descendant. Similarly, if e

is itself unlabeled, and has no labeled ancestor, then e is accessible if and only if it has an accessible descendent.

At this stage, we know that node e is unlabeled. We also know that its nearest labeled ancestor, if one exists, say g, has a $d-$ label. Consider a node f that is a nearest labeled descendant of e. (That is, there are no labeled nodes intervening between e and f). (There could be multiple such nodes, we choose f to be any one of them). If no such f can be found, then e has no accessible descendants, and we are done, and conclude e is inaccessible.

If there is such an f and g, then g is also the nearest labeled ancestor of node f (since e is unlabeled). There are three possible labels for node f: $(d+, s+)$, $(d-, s+)$, and $(d-, s-)$. On account of g, the last of these labels is redundant at f. As such, it cannot occur in any optimal CAM, and the label on f must be one of the other two, in both of which f has $s+$. Therefore, e is accessible.

If there is such an f but no labeled ancestor g, then all ancestors of f, including e, must be upward redundant. (The only way for an ancestor-free node to become unlabeled is for its label to be upward redundant). Therefore, e is accessible irrespective of the label on f. □

By linking the nodes in the trie representation of the CAM back to the corresponding nodes in the database tree, the CAM can also be used to enumerate, for a given user and access type, the list of all data nodes accessible by the user. This can be achieved in time proportional to the size of the output plus the size of the CAM, but independent of the size of the original database tree.

6. MULTIPLE UNIT REGIONS

Having addressed the case where the marked database tree consisted of a single unit region, we now turn to the the general case of multiple unit regions. A database tree has more than one unit region exactly when it contains an inaccessible node with an accessible child. Call such inaccessible nodes *inter-region terminal* nodes, or IRT nodes for short. When such nodes are present, one could introduce a fourth label $(d+, s-)$ to mean that the node carrying this label is inaccessible, but its descendants are accessible unless overridden by other labels "close" to them. However, one loses the ancestor accessibility property, and the compression that it affords. Instead, we show here how to utilize the results of the preceding sections and exploit the ancestor accessibility in the unit regions where it holds.

Recall, from Section 2.3, the notion of *marker nodes*, introduced to mark off regions of a database tree where the ancestor accessibility property holds locally: a marker node is an accessible child of an inaccessible node. Also recall that the unit region is a maximal subtree of the database tree that is rooted at either the tree root or at any marker node and excludes any marker descendants of the root of the unit region, as well as their descendants. We are interested in labelings that respect certain constraints.

(1) $(d+, s+), (d-, s+), (d-, s-)$ are the only labels allowed, and
(2) Every marker node has to be identified.

We call any labeling that respects these constraints a *constrained labeling*. The size of a constrained labeling is the number of nodes that are either labeled or

marked by it. We seek optimal constrained labelings for database trees with multiple unit regions. We have the following proposition.

PROPOSITION 6.1. *(Inter-Region Terminal Label) An inter-region terminal node is never labeled in an optimal constrained labeling.*

PROOF. A constrained labeling always identifies marker nodes. A node is an IRT node exactly when it has a marker child, so we can always locate IRT nodes. The only possible label an IRT node can have in any constrained labeling is $(d-, s-)$. But then this label can be inferred from the identity of IRT nodes, which is available in any constrained labeling. □

We have the following straightforward algorithm for constructing constrained labelings for database trees with multiple unit regions. First, perform a decomposition of the database tree H into unit region components. Reduce each unit region by deleting the subtree rooted at each of its inter-region terminal nodes. Second, using Algorithm Opt-UnitRegion-CAM label each reduced unit region individually. Finally, take the union of these reduced unit region labelings, and mark each marker node. Call this modified algorithm Algorithm Opt-MultiRegion-CAM.

We can show that this algorithm always yields an optimal constrained labeling. A key lemma that needs to be established first is that if there is an optimal labeling of a unit region that assigns a label to its root, then Algorithm Opt-UnitRegion-CAM will produce such a labeling. The motivation is the following. We require marker nodes to be marked by all constrained labelings of the database tree. Suppose there is an optimal labeling that labels some (reduced) unit region root but the one produced by Algorithm Opt-UnitRegion-CAM does not label that root. Then the overall labeling obtained from Algorithm Opt-MultiRegion-CAM would mark an additional unlabeled node whereas in the supposed optimal labeling, we mark a labeled node. Marking a labeled node can be accomplished with one bit whereas marking a new (unlabeled) node costs additional space needed to identify the node. Thus, the overall labeling obtained from Algorithm Opt-MultiRegion-CAM would be suboptimal.

Before establishing this key lemma, we need a syntactic characterization of when a node (label) is upward redundant w.r.t. the optimal labeling produced by Algorithm Opt-UnitRegion-CAM.

We say an internal non-terminal node e in a marked single unit region database tree H is *covered* provided e is accessible and every child c of e satisfies one of the following conditions:

—c is an internal terminal node.

—c is covered.

In the following proof, we refer to a node as a *signed* node provided it is either positive or negative (see Definition 4.8). Otherwise, we call is *neutral*.

LEMMA 6.2. *(Characterization of Upward Redundancy) An internal node of a single unit region database tree is upward redundant w.r.t. the optimal labeling produced by Algorithm Opt-UnitRegion-CAM iff it is covered.*

PROOF. Let L_{opt} denote the labeling produced by Algorithm Opt-UnitRegion-CAM.[5]

($\Leftarrow$): Suppose every child of e is internal terminal. Since L_{opt} labels every internal terminal node, e is clearly upward redundant. Inductively, assume that whenever a covered node has no more than $k-1$ proper covered descendants. Suppose e has k proper covered descendants. Consider any child c of e. Clearly, c has no more than $k-1$ proper covered descendants and is thus upward redundant by induction hypothesis. From Definition 3.6 it follows that e is upward redundant as well.

($\Rightarrow$): Suppose e is upward redundant w.r.t. L_{opt} but is not covered. Let e be the deepest internal non-terminal node of H that is not covered. Note that e must have at least one signed child. Stage I of Algorithm Opt-UnitRegion-CAM will assign some label $(d*, s+)$ to e, where $d*$ may be $+$ or $-$. This label will subsume the label of every positive child of e if $*$ is $+$ (every negative child if $*$ is $-$). Since rule 2 is always applied prior to rule 1' by Algorithm Opt-UnitRegion-CAM, e will not be upward redundant. Suppose e has one or more proper descendants that are internal, non-terminal, and not covered. We can show by induction that at least one of e's children will not be upward redundant w.r.t. L_{opt} whence e is not upward redundant as well. □

It is easy to see now that in a single unit region database tree, a node is upward redundant w.r.t. the optimal labeling produced by Algorithm Opt-UnitRegion-CAM iff it is so w.r.t. any optimal labeling. The reason is that if a node is covered then it will become upward redundant w.r.t. every optimal labeling.

We next prove the key lemma.

LEMMA 6.3. *(Favoring Root) Consider a database tree H with a single unit region. Then if there is any optimal labeling in which H's root is labeled, Algorithm Opt-UnitRegion-CAM will produce such a labeling.*

PROOF. Let L_{opt} be the labeling produced by Algorithm Opt-UnitRegion-CAM. The only circumstance under which L_{opt} leaves the root unlabeled is that the root is upward redundant w.r.t. L_{opt}, or equivalently, the root is covered. We will show by induction on the number k of covered proper descendants of the root that no optimal labeling will label the root. Let L be any optimal labeling of H.

Base Case: $k = 0$. In this case, all children of the root must be internal terminal nodes.

Let L_{opt} be the optimal labeling produced by Algorithm Opt-UnitRegion-CAM. The only circumstance under which L_{opt} leaves the root unlabeled is that every child of the root is either internal terminal or is upward redundant (w.r.t. L_{opt}). We shall show that no optimal labeling could have the root labeled in this case. Let L be any optimal labeling of H. If L labels all children of the root, any label on the root will be redundant, since the child labels must all be $(d-, s+)$. Suppose a child c of the root is unlabeled in L. Then L must label the root $(d+, s+)$ so c's accessibility can be determined and it must also label every child of c $(d-, s-)$.

[5]Sometimes, the root is arbitrarily assigned one of $(d+, s+)$ or $(d-, s+)$. The upward redundant status of a node does not depend on this choice and we freely speak of *the* optimal labeling produced by the algorithm.

The size of L in this case will be strictly greater than that of the optimal labeling produced by Algorithm Opt-UnitRegion-CAM.

Induction: $k > 1$.

Consider any child c of the root. The number of proper covered descendants of c must be $< k$. If c is internal terminal, then L must label it $((d-, s+))$ by virtue of an argument similar to that for the base case. Otherwise, by induction hypothesis, we have that L will not label c, i.e. c is upward redundant w.r.t. L. So every child of the root is either internal terminal or is upward redundant so, by Definition 3.6, it follows that the root itself is upward redundant w.r.t. L in which case, L will not label it. □

In comparing the sizes of constrained labelings, we charge for the number of nodes that are either labeled or marked. In particular, if a node is both marked and labeled, it is counted only once.[6] We now have the following theorem.

THEOREM 6.4. *(Optimality) Let H be any marked database tree, possibly with multiple unit regions. The CAM obtained by applying Algorithm Opt-MultiRegion-CAM is of the smallest size among all constrained labelings of H.*

PROOF. Let L be any optimal constrained labeling of H. Consider L restricted to each unit region of H. We will show that the optimal labeling, say L_{opt}, produced by Algorithm Opt-MultiRegion-CAM has a size no larger than L in the unit region, from which the theorem will follow. By definition, L must mark each marker node. So it suffices to consider labeled nodes when comparing sizes, as long as both labelings label (or both do not label) the marker nodes. We can ignore any inter-region terminal nodes since such nodes are never labeled in L_{opt}. If the root of the unit region is labeled in L, then it follows from Lemma 6.3 that L_{opt} would label the said root as well. From Theorem 4.11, the size of L_{opt} restricted to this unit region is no more than that of L restricted to it. Suppose L does not label the root of the unit region. Once again, it follows from Theorem 4.11 that L_{opt} restricted to the unit region is no larger than L restricted to it. This was to be shown. □

CAM lookup for a database tree with multiple unit regions is quite similar to the case for a single unit region, and is presented in Algorithm LookUp-MultiRegion-CAM, in Figure 7. The key difference with Algorithm LookUp-UnitRegion-CAM is that the lookup algorithm for multiple regions needs to account for marker nodes defining unit region boundaries. Specifically,

Specifically, our main lookup theorem is now restated as:

THEOREM 6.5. *(Multi-Region CAM Lookup) Given a CAM corresponding to a multi-region database tree, Algorithm LookUp-MultiRegion-CAM correctly determines whether a specified node is accessible. Further, it does so in time proportional to the product of the depth of the node in the XML tree and logarithm of the CAM size.* □

[6]It costs one extra bit to mark a labeled node, but this can safely be neglected. What is important is that it costs more to mark an unlabeled node.

```
Algorithm LookUp-MultiRegion-CAM
  // given a CAM trie C, and a query node e in database H
  let f1 in C denote the nearest labeled ancestor-or-self of e (in H);
  let f2 in C denote the nearest labeled descendant (if any) of e (in H);
  let f3 in C denote the labeled node sharing the longest common prefix with e;
  if (f1 = e) return (e's accessibility as determined by the s* label of f1);
  else if (label of f1 is (d−, s−)) return (e is inaccessible);
  else if (label of f1 is (d+, s+))
     if ((f3 is a marker node) and (e is a descendant-or-self of the parent of f3))
        // e is a descendant of an IRT node
        return (e is inaccessible)
     else return (e is accessible)
  else // either f1 does not exist or label of f1 is (d−, s+)
     if ((there is no f2) or (f2 is a marker node))
        return (e is inaccessible);
     else return (e is accessible); // by ancestor accessibility
```

Fig. 7. Looking up accessibility of node e using a multi-region CAM trie

7. EXPERIMENTAL VERIFICATION

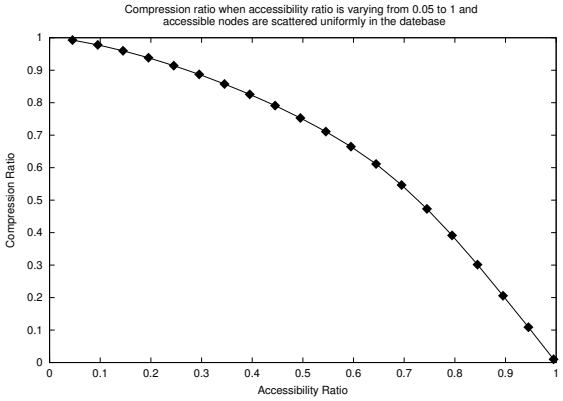
To validate the utility of a compressed accessibility map, we ran a variety of experiments that are intended to be indicative of how our techniques would perform for complex XML data that specifies access control information. We could not find a large production XML system with well-defined access control information; hence, we used synthetic XML data. However, hierarchical file systems (with real access control data) were readily available. Although our techniques are designed for supporting efficient access control for XML data, we made the (reasonable) assumption that the nature of access locality in XML data would be similar to that in hierarchical file systems. It is important to note, though, that our techniques do not make any assumptions about how the XML documents are stored.

7.1 Experiments with Synthetic XML Data

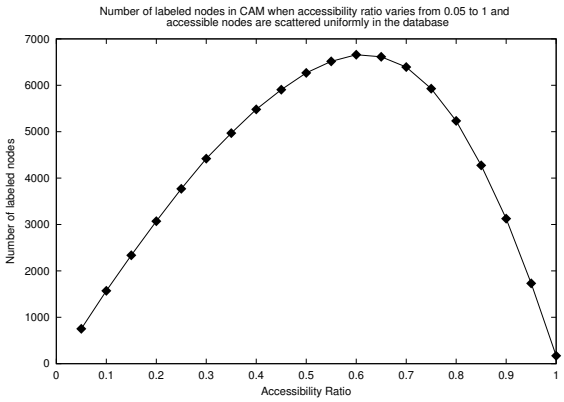
To show clearly how access locality affects the compression ratio, we experimented with synthetic XML data.

The Data. We used XML documents generated using IBM AlphaWorks' XML generator [Diaz and Lovell 1999]. Several parameters are provided by the XML generator that can be used to control the structure of the generated XML documents (e.g., fanout and depth). We ran our experiments for a variety of XML documents and obtained essentially similar results. We report here results for a large representative XML tree, with 16811 nodes, with variable fanout (maximum = 60, minimum = 1, average = 2, not including the leaf nodes), and variable distances between leaves and root (average depth of 8), for different types of access locality.

No Access Locality. We first dealt with the case where accessible nodes are uniformly randomly distributed in the XML data. Note that access locality is the worst in this case. Figures 8(a) and 8(b) show the compression ratio and space



(a) Compression Ratio versus Accessibility Ratio



(b) Space cost of CAM versus Accessibility Ratio

Fig. 8. Accessible nodes are uniformly distributed

cost of the CAM, respectively, when the accessibility ratio is varied from 0.05 to 1. (The *accessibility ratio* is the fraction of nodes in the database that are accessible.)

From Figure 8(a), we can see that the larger is the accessibility ratio, the better is the compression ratio achieved by the CAM. The reason is that, while the space requirement of the fully materialized accessibility map grows linearly with the accessibility ratio, the space requirement of the CAM grows much more slowly. When the accessibility ratio is small, the fully materialized accessibility map requires less space. Therefore, even though the CAM does not require much space, the compression ratio is quite high. On the other hand, when the accessibility ratio is large, the fully materialized accessibility map requires far more space than the CAM, which makes the compression ratio much better. In particular, when the accessibility ratio is close to 1, accessible nodes tend to be close to each other, which results

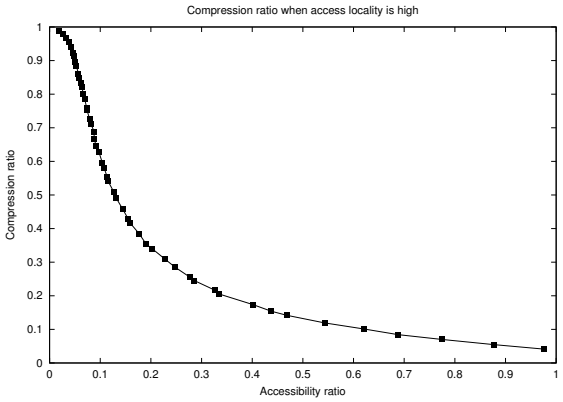


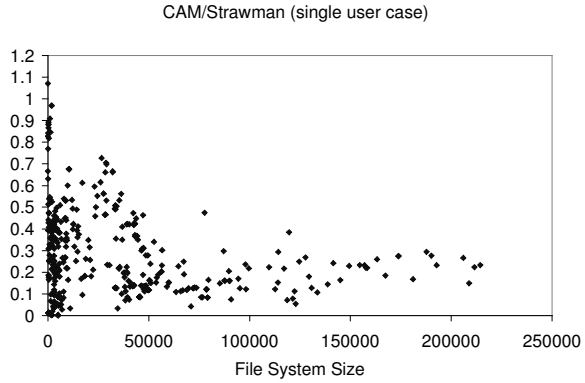
Fig. 9. Compression ratio versus accessibility ratio when locality is high

in better locality. In this situation, because of the way CAM takes advantage of access locality, it takes very little space.

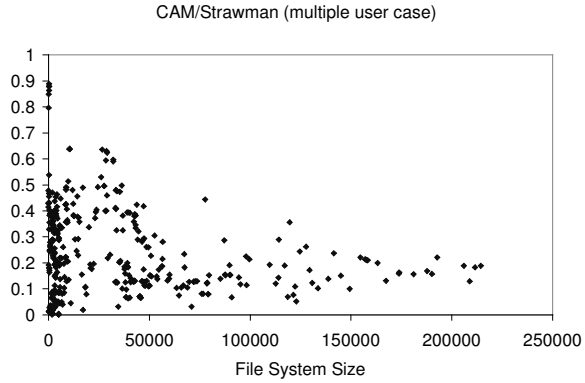
Varying Access Locality. Here, we present experimental results for the case when the database has better access locality. In our synthetic XML data, a node is called *friendly* (resp., *non-friendly*) if the objects in the subtree rooted at that node have a high (resp., low) probability of being accessible. We call the subtree rooted at a friendly node (resp., non-friendly node) a *friendly area* (resp., *non-friendly area*). Two parameters of interest are af and anf, which define the access probability of a node in a friendly area and in a non-friendly area, respectively. Initially, we set the root to be a friendly node. To allow for multiple unit regions, we use two additional parameters. The *friend ratio*, or fr, is the probability that a node is a friendly node given that its parent is a non-friendly node. Similarly, the *reverse ratio*, or rr, is the probability that a node is a non-friendly node given that its parent is a friendly node.

To have better locality, fr should be small which will result in few unit regions in the whole database. In our experiment, we set fr to be around 0.05. af and anf are set to be 0.98 and 0.02 respectively. In order to get various accessibility ratios, we vary rr from 0 to 1. However, accessibility ratios are also affected by fr. We find that if fr is fixed to be 0.05, the smallest accessibility ratio we can get is only around 0.1 (when $rr = 1$). To achieve even smaller accessibility ratios, fr needs to be decreased accordingly. In the experiment, we set $fr = 0.06 - \frac{0.06}{21-20rr}$. The idea is to let fr be around 0.05 for most values of rr. When rr is close to 1, fr is decreased gradually so that we can get accessibility ratios close to 0.

Figure 9 shows the compression ratio when the accessibility ratio changes from 0.02 to 0.98. It is quite clear that when the accessibility ratio is very small (around 0.1), the compression ratio achieved by the CAM is not good. In fact, when the accessibility ratio is close the 0, the trend is similar to the case where accessible nodes are uniformly randomly distributed in the XML data. However, as the accessibility ratio increases, the compression ratio improves dramatically. Even for a relatively low accessibility ratio like 0.25, the compression ratio is still very impressive. Compared with Figure 8(a), we see clearly how access locality greatly affects



(a) Single user



(b) Multiple users

Fig. 10. Compression ratio versus file system size

the compression ratio that the CAM can achieve.

7.2 Experiments with Real File System Data

The Data. We obtained access control information on 433 UNIX file systems at a large university. Our sample included a variety of file systems, ranging from file systems on "old" servers, with faculty users who had been with the University a long time, to new file systems on machines purchased recently. The file systems included the personal systems of a variety of users, students, visitors, and faculty, as well as data associated with several large data intensive projects and much system software.

The Metrics. To demonstrate the space-efficiency of our CAM approach, for each file system, we compare the storage cost of the CAM with that of the fully mate-

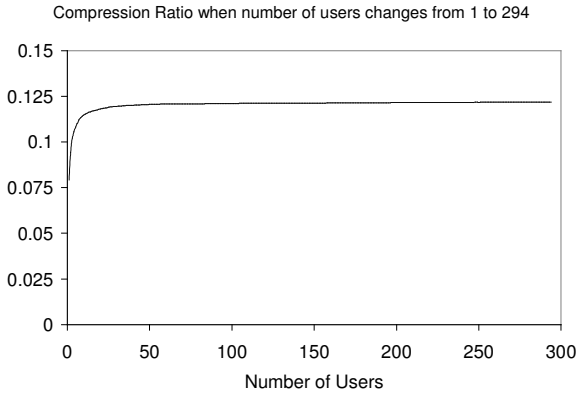


Fig. 11. Compression ratio versus # of users

rialized accessibility map, for multiple users. In the CAM approach, three CAMs (for read, write and execution permissions respectively) are maintained for each user. Thus, the total space cost of the CAM approach is the sum of the sizes of the CAMs of each user. In the fully materialized accessibility map, each object in the database maintains an access control list for each permission, which lists user IDs of the users who have read/write/execution access to that object, based on the standard UNIX semantics. Therefore, the space cost is the sum of the sizes of these lists over all the objects. Given a user set U, the *compression ratio* is calculated as the total space cost of the CAM approach for U over that of the fully materialized accessibility map.

Compression Ratio Results. First, we examined the compression ratio for a single user. Figure 10(a) shows the compression ratio versus file system size for the worst of twenty users that we tested. We can see that when a file system is reasonably large (over 25,000 files), in most cases the compression ratio is quite good (around 0.2). Also, it is fairly clear that most cases of poor compression were for small file systems. Thus, one observes a trend towards better compression ratios as file system size increases.

Figure 10(b) presents compression ratio versus file system size when U is set to be all the users of a file system. It is quite heartening that the chart of compression ratio for the multi-user case is very similar to that of the single user case in both shape and trend. The reason is that most users have access to only a small portion of the file system. Therefore they tend to have similar compression ratios which makes the overall compression ratio for multiple users similar to that of a single user.

To illustrate this point more clearly, we ran another experiment on one file system which is of size 78501 files and with 294 users. Figure 11 shows how the compression ratio varies as the size of U increases. The set U is ordered by traversing the file system depth first, and adding to U each new user found. In the beginning, most users in U are superusers (who tend to have ownership of the file system root and its immediate descendants). Therefore, the compression ratio is very good. As more and more ordinary users are added to U, the compression ratio goes up to

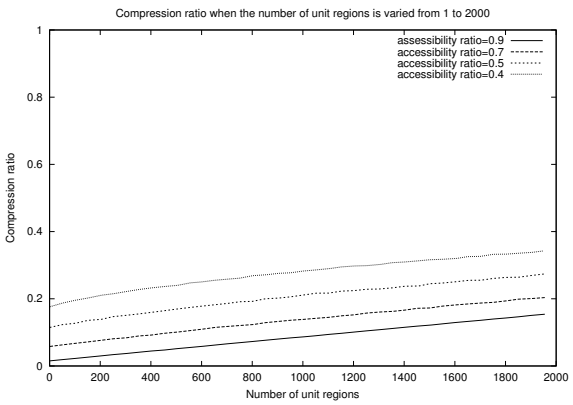


Fig. 12. The effect of multiple regions

around 0.12 and remains constant even when U's size increases.

Another interesting observation is that the worst case behavior for a single user (shown in Figure 10) is for the user login "guest". This is because the access locality for a guest user (files with read permission for "other") is poorer than that of an ordinary user. By locality, we mean that the accessible objects tend to be located in one or more subtrees instead of being randomly scattered throughout the whole file system.

7.3 The Effect of Multiple Unit Regions

We also ran experiments over synthetic data to study how the number of unit regions in an XML database will affect the compression ratio that CAM can achieve. Our previous experiment results (Figures 8(a) and 9) show that CAM's compression ratio partially depends on the accessibility ratio of the database. In order to demonstrate clearly the effect of multiple unit regions, it is desirable to only vary the number of unit regions while keeping the accessibility ratio relatively constant. To achieve this end, we introduce a parameter *propagation ratio* (*pr* for short) in the experiments, which controls the probability that a node is accessible when its parents is accessible. Given an XML database and the desired number of unit regions k, we first randomly select k nodes as marker nodes. Then in each unit region, we assign each node's accessibility according to *pr*. By this method, two databases with different number of unit regions will have relatively the same accessibility ratio as long as the same *pr* is used. Note that if the number of unit regions are comparable to the size of the XML database, the accessibility ratio will inevitably increase when number of unit regions increases. Therefore, we ran the experiments over a database with size 20000 and vary the number of unit regions from 1 to 2000. Figure 12 shows the compression ratio under different accessibility ratios. We can see that with the same accessibility ratio, the more unit regions a database has, the bigger the compression ratio CAM achieves. This agrees with our intuition that better locality (i.e., fewer unit regions) yields better compression ratio. Notice that because of the high locality of accessibility in the dataset, even when the accessibility ratios are pretty low, the corresponding compression ratios

are still very satisfactory. It is also interesting to observe that the compression ratio for each accessibility ratio increases linearly almost at the same rate along with the number of unit regions, which suggests that the impact of multiple unit regions is orthogonal to that of accessibility ratio.

8. INCREMENTAL CAM MAINTENANCE

It is expected that new nodes will be added to the XML database frequently, and existing nodes deleted. There will be corresponding changes to the list of nodes accessible to a user. The bulk of these changes involves the creation or deletion of a leaf node. Occasionally, access control policies, or the properties or categories of a user may change, resulting in a change to the set of objects accessible to a user. The question is how to maintain the CAM so that it continues to be optimal after the change. One could always re-run the optimal labeling algorithm. Even though it takes time linear in the database size, it is worth asking whether we can do better, particularly for large databases.

It is easy to see that, in the worst-case, a single small change can cause a wholesale re-labeling of an entire database tree. For example, consider a database tree as follows: an accessible root A, with n accessible and n inaccessible leaf children, and one accessible internal child B; node B in turn has n accessible and n inaccessible leaf children and one accessible internal child C; and so on until some internal node X in this chain has just n accessible and n inaccessible leaf children. One optimal labeling has $(d+, s+)$ at the root A, and a label $(d-, s-)$ at each inaccessible leaf. All other labels are removed. Now suppose one inaccessible leaf child is added to node X. The unique optimal CAM now has a label of $(d-, s+)$ at the root A, and a $(d+, s+)$ label at each accessible leaf. All inaccessible leaf labels are removed. In other words, *every* node in the tree has a label change due to the single addition of one leaf child. Clearly, the size of this tree could be arbitrarily large: we could set n as large as we wished to increase fanout, and make the chain of internal nodes from A to X as long as we wished to increase depth. This leads to the theorem below.

We say that two marked database trees H and H' differ on a node provided either (i) exactly one of them has that node, or (ii) both H and H' have that node, but differ on its mark.

THEOREM 8.1. *(Cost of Incremental Change) For every n, there are marked database trees H and H', differing on exactly one node, such that their optimal CAMs differ on at least n nodes.* □

In light of this theorem, there is little hope for preserving optimality with a limited amount of incremental work. However, it turns out that we can come very close, as we show below. To do so, we begin by establishing some notions with respect to small changes.

Definition 8.2. (Local Consistency) A label ℓ for a node e in H is locally consistent with its marking M provided:

—e is accessible according to M and $\ell = (d*, s+)$; *or*

—e is not accessible according to M, and $\ell = (d-, s-)$.

Here, $*$ may be $+$ or $-$. □

Local consistency at e means it is possible to find labels for the rest of the nodes in the tree so together they will respect M. Let label ℓ at node e be locally consistent w.r.t. M. The constraint $label(e) = \ell$ is satisfied by a complete labeling provided it assigns this label to e. A CAM for a database tree with marking M satisfies this constraint provided it is a CAM associated with any complete labeling that satisfies this constraint. An optimal constrained CAM (for this constraint) for a tree with given marking M is any CAM which satisfies this constraint and is of the smallest size among all such CAMs.

The next two theorems show that optimal CAMs for database trees can be incrementally maintained in an efficient way, for a very small price. As long as we are willing to permit one label more than the optimal, we can add or delete an entire labeled subtree in $O(1)$ time. The labeling of the subtree itself takes time proportional to its size, using the optimal labeling algorithm. Thus, an entire subtree of new nodes can be added to an existing database, with the near-optimal CAM incrementally maintained in time proportional to the size of the subtree, and independent of the size of the database. Where several updates are made in structural proximity, a suggested incremental maintenance technique is to batch these updates, identify a local subtree that includes all these updates and completely recompress it, and finally pay a penalty of at most one extra label for (re-)attaching this subtree to the original CAM. With such batching, near-optimal incremental maintenance can continue for long periods before there is a need to re-compress the entire database.

LEMMA 8.3. *(Local-Change-Count) The change of a label at a node e, locally consistent with a given marking, can change the labels below e, in an optimal CAM, but the number of labels is the same except at e and its immediate children.*

PROOF. The only change we need to consider is from $(d-, s+)$ to $(d+, s+)$, or the reverse. From the proof of optimality of our labeling algorithm, we know that this change to the label at a node e should not affect the (possibly redundant) labels applied to any of its descendants, except those with deferred labels. But all such labels are eventually subsumed. Which labels below these deferred label nodes get subsumed is a function of how the deferred label is resolved, but the total number of these is identical in either case, by construction (of the label deferral). Only at immediate children of e can there be a difference in count on account of some labels being subsumed and others not, depending on the label chosen for e. □

The following two lemmas are used to prove the key results about preserving near-optimality incrementally.

LEMMA 8.4. *(Positive Change to Label) Let H be a marked database tree and K a complete labeling for H such that for some node e, $label^K(e) = (d-, s+)$. Let $\mathcal{I}$ be an optimal CAM associated with K. Let $\mathcal{I}'$ be any optimal CAM equivalent to $\mathcal{I}$ except it satisfies the locally consistent constraint $label(e) = (d+, s+)$. Let H_e be the tree H with the subtree rooted at e deleted. Then $\mathcal{I}'$ has at most one more labeled node than $\mathcal{I}$, in H_e.*

PROOF. Suppose K' is the complete labeling for H whose associated optimal CAM is $\mathcal{I}'$. So, $K'(e) = (d+, s+)$. Let f be the parent of e.
Case 1: f had the label $(d+, s+)$ in K.

Then changing e's label to $(d+, s+)$ (to satisfy the constraint) does not affect f or the rest of the nodes in H_e, in the constrained optimal CAM $\mathcal{I}'$. The label of e itself is now subsumed in K', whereas previously, in K, it was not. So, in this case, in the subtree of H_e rooted at f the number of labeled nodes does not increase from $\mathcal{I}$ to $\mathcal{I}'$, while in the rest of the tree H_e, there is no change.
Case 2: f originally had the label $(d-, s+)$ in K, and continues to be so labeled in K'.

Then the change of label at e does not affect f (by case assumption) and hence nor the rest of the nodes in H_e in the CAM $\mathcal{I}'$. In H, the label of e itself is now non-redundant in K', whereas it previously may have been redundant in K. So, in H_e, $\mathcal{I}'$ has exactly the same number of labeled nodes as $\mathcal{I}$.
Case 3: f was originally labeled $(d-, s+)$, and is now labeled $(d+, s+)$ in K'.

Then, first off, notice that the label at e is now redundant in K', and may previously have been redundant in K as well (depending on whether or not e is a non-leaf terminal node). Several sub-cases to consider. Suppose there were an equal number of positive and negative children of f before the change, and now there are more positive children. Then the number of labeled children of f in $\mathcal{I}'$ does not increase. Suppose there were more negative children before the change, and there are more positive children now, then again the number of labeled children does not increase. Suppose there were more negative children before the change, and there are an equal number k of positive and negative children now in K'. We then have non-redundant labels at these k negative children now, whereas previously, in K, we had non-redundant labels at $k-1$ positive children of f. However the label at f itself is now redundant, whereas it previously was not. This is because f's label now is deferred and hence subsumed, whereas it wasn't so in K. So once again, the number of labels in the subtree of H_e rooted at f does not increase.

To sum, in Cases 1 and 2, the change to e's label does not propagate any label change to e's parent, where as in Case 3, such a change ripples up. To complete the proof, we apply induction. If e's parent has its label changed in K' and hence in $\mathcal{I}'$, apply the arguments in Cases 1-3 inductively. A key point to note is that whenever there is a ripple, the number of labeled nodes of $\mathcal{I}'$ in the subtree of H_e rooted at e's parent (for the "current" e) does not increase compared to $\mathcal{I}$. On the other hand, when the above number goes up (always by at most 1), there is no ripple. The induction thus terminates whenever a parent node's label does not change (Case 1 or 2), or when the ripple reaches the root. In the latter case, the number of labeled nodes of $\mathcal{I}'$ in H_e for the given e is exactly the same as for $\mathcal{I}$. If the ripple stops midway, the number will be up by at most one. The lemma follows. □

LEMMA 8.5. *(Negative Change to Label) Let H be a marked database tree and K a complete labeling for H such that for some node e, $label^K(e) = (d+, s+)$. Let $\mathcal{I}$ be an optimal CAM associated with K. Let $\mathcal{I}'$ be any optimal CAM equivalent to $\mathcal{I}$ except it satisfies the locally consistent constraint $label(e) = (d-, s+)$. Let H_e be the tree H with the subtree rooted at e deleted. Then $\mathcal{I}'$ has at most one more*

labeled node than $\mathcal{I}$, in H_e.

PROOF. Follows in a manner similar to the proof of Lemma 8.4. □

THEOREM 8.6. *(Subtree Addition) Let H_1 and H_2 be any two marked database trees and H be the tree obtained by making the root r of H_1 a child of some accessible node e in H_2. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be the optimal CAMs for H_1 and H_2, obtained using our algorithm and let $\mathcal{I}$ be the CAM for H, obtained by taking $\mathcal{I}_1 \cup \mathcal{I}_2$, and deleting the label of r if one exists and is subsumed (in $\mathcal{I}_1 \cup \mathcal{I}_2$). Suppose K_{opt} is the optimal CAM for H obtained directly using our algorithm. Then $\mathcal{I}$ has at most one more labeled node than K_{opt}.*

PROOF. **Case 1:** Node r has unequal numbers of positive and negative children, or is a leaf.

Consider the way nodes are assigned labels in the construction of K_{opt}. Since labeling proceeds bottom up in the optimal algorithm, nodes in H_1 must be labeled identically in K_{opt} and $\mathcal{I}_1$.

Next, compare K_{opt} with $\mathcal{I}_2$ for nodes in the subtree H_2 of H. Here, K_{opt} and $\mathcal{I}_2$ assign labels to nodes identically from leaves up to node e, where they might differ. Suppose e is labeled alike in both. Then it is easy to show, by induction, that all other node labels in the rest of H will coincide in both. So suppose they differ on e. Then by Lemmas 8.4 and 8.5, in the subtree H_e of H obtained by deleting the subtree rooted at e, $\mathcal{I}$ has at most one more label than K_{opt}. So now focus on the subtree of H rooted at e. If e was a leaf of H_2, then we are done. Else w.l.o.g., let the label of e in $\mathcal{I}_2$ be $(d-, s+)$ and in K_{opt} be $(d+, s+)$. This implies that r has a label of $(d+, s+)$. The following subcases arise.

Case 1.1: e has an equal number of positive and negative children in H_2 (i.e. r is not counted), and the optimal label assigned to e after deferment is $(d-, s+)$ in $\mathcal{I}_2$.

So e has a subsumed label in $\mathcal{I}$ but not in K_{opt}. On the other hand, r must have a subsumed label in K_{opt} (since r forces a label on e in K_{opt}) but not in $\mathcal{I}$. These cancel each other out in counting the number of labeled nodes in the two CAMs. Next, consider the other children of e: whereas the negative children of e are subsumed in $\mathcal{I}$, it is the positive children that are subsumed in K_{opt}. Since these numbers are equal (recall r is already dealt with above and is not counted here), the changes cancel each other out. Considering other descendants of e, there is no difference between the two CAMs in the count of labels, by Lemma 8.3.

Case 1.2: e has one negative child more than positive children, not counting r, so that the optimal labeling of e in $\mathcal{I}_2$ is $(d-, s+)$.

In this case, in H, e will have an equal number of positive and negative children. Its label is deferred and is resolved to $(d+, s+)$. Clearly, this label is subsumed in K_{opt} whereas in $\mathcal{I}$ it is not. Besides, in K_{opt}, the labels of e's positive children, including r are also subsumed, whereas in $\mathcal{I}$ it is only e's negative children that are subsumed. Among e and its children in H, $\mathcal{I}$ has one more label than K_{opt}. Considering other descendants of e, there is no difference between the two CAMs in the count of labels, by Lemma 8.3.

For the rest of tree H, by the arguments made in the proof Lemma 8.5, we see that the label at the parent of e could not possibly have changed, and as such, there is no additional changes at ancestors to consider.

Case 2: Node r has an equal number of positive and negative children.

The label at r could be different in $\mathcal{I}$ and K_{opt}. But by arguments similar to those made in the preceding case, we can show that the number of labels in the proper descendants of r remains the same in the two CAMs. The label at r itself may or may not be subsumed in $\mathcal{I}$, whereas it is definitely subsumed in K_{opt}, leading to a difference of at most 1. Finally, the label at e, and hence that of all nodes in H_2, remains the same between the two CAMs, since r is neither positive nor negative, and hence does not affect the label choice at e. Once again, the overall difference between is at most one label.

To conclude the proof, we need to account for upward redundancy of e so far. However, it is straightforward to see that e is upward redundant in H_2 iff it is in H. The theorem now follows. □

THEOREM 8.7. *(Subtree Deletion) Let H' be any database tree and H the tree obtained by deleting the subtree of H' rooted at some node r. Let $\mathcal{I}'$ be the optimal CAM for H' and $\mathcal{I}$ its restriction to H. Furthermore, let K_{opt} be the optimal CAM for H. Then $\mathcal{I}$ has at most one more labeled node than K_{opt}.*

PROOF. Let e be the parent of r in H'. e is retained after the subtree deletion, whereas r is removed. If the label of e computed by the optimal algorithm remains the same whether or not the subtree rooted at r is present, then there is no change to any other labels in the retained tree H. In this case, $\mathcal{I}$ is identical to K_{opt}. So assume that the label on e is affected by the removal of r (in K_{opt}). There are two such cases: the label at r is $(d+, s+)$, or the label at r is $d-$ (with an $s-$ or a non-terminal $s+$). We prove the former case. The latter case follows a similar argument. There are two possibilities within the former case.

Case 1: Removal of r leaves e with an equal number of positive and negative children in H.

In this case, e must have had one positive child more than it has negative children in H' and hence must be labeled $(d+, s+)$ in $\mathcal{I}'$ as also in $\mathcal{I}$. In K_{opt}, e must have its label deferred. The only non-trivial case is that e's label is ultimately resolved to $(d-, s+)$ in K_{opt}. Note that there is no difference between $\mathcal{I}$ and K_{opt} in the number of labels below e in the tree H. The label at e itself is subsumed in K_{opt}, but may or may not be subsumed in $\mathcal{I}$.

If the label at e is not subsumed in $\mathcal{I}$, then the label of e's parent must be the same in $\mathcal{I}$ and in K_{opt}. In the terminology of Lemmas 8.4 and 8.5, the change of labels from $(d-, s+)$ to $(d+, s+)$ does not ripple up beyond e. From those lemmas, it follows that $\mathcal{I}$ then has exactly one more label than K_{opt}.

If the label at e is subsumed in $\mathcal{I}$, then the label at e's parent is different in $\mathcal{I}$ and in K_{opt}. In this case, note that in the subtree of H rooted at e, $\mathcal{I}$ has exactly as many labels as K_{opt}. By Lemmas 8.4 and 8.5, $\mathcal{I}$ has at most one more label than K_{opt} in the subtree H_e of H obtained by removing the subtree rooted at e. Overall, $\mathcal{I}$ has at most one more label than K_{opt}.

Case 2: e has an equal number of positive and negative children in H'.

In this case, r's removal will force e to be labeled $(d-, s+)$ in K_{opt}, whereas e's label is deferred in $\mathcal{I}$. The only interesting case is it is ultimately resolved to $(d+, s+)$. In this case, e's label is subsumed in $\mathcal{I}$ whereas it is not in K_{opt}. On the other hand, $\mathcal{I}$ has an extra label below node e compared to K_{opt}. (We only need to

count immediate children of e on account of Lemma 8.3). Since $\mathcal{I}$ and K_{opt} differ on the label they assign to e (although this label is subsumed in $\mathcal{I}$), once again Lemmas 8.4 and 8.5 apply, from which it follows that in the subtree H_e (and hence in H), $\mathcal{I}$ has at most one more label than K_{opt}. □

9. RELATED WORK

Commercial relational DBMSs support "fine grained" (i.e., per data item) access control, and there is considerable literature on the subject.

9.1 Access Control Techniques

Access control techniques can broadly be divided into mandatory control techniques and discretionary control techniques. Mandatory access control (see, e.g., [Lunt et al. 1990; Jajodia and Sandhu 1991; Sandhu 1993; Winslett et al. 1994]) is based on associating *security labels* with each data item and user. A label associated with a data item is called a security classification, while a label associated with a user is called a security clearance level. A user is allowed access to a data item only if the user's clearance level is at least as high as the data item's classification. Most mandatory techniques tend to be relatively simple, and use some kind of belief logic-based semantics, where a secure database corresponds to a set of databases: user s's database has precisely those tuples that s believes hold w.r.t. its clearance level. A CAM can be used to capture this view efficiently, if data items have a suitable hierarchical organization imposed on them.

The idea behind discretionary access control is to permit flexible specifications of which users can access which data items using, for example, access control lists. Database systems have tended to lean more towards discretionary access control. The history of work on discretionary access control begins as early as System R [Griffiths and Wade 1976; Fagin 1978]. Notable extensions and improvements include the ORION model [Rabitti et al. 1991], a model due to Gal-Oz et al. [1993], and the model of Bertino et al. [1997]. Access control is effected via views that are defined in the native query language of the database system. The view definitions may be used to explicitly define permissions or denials, depending on the type of "meta-policy" (such as "denials-take-precedence") one wants to adopt. The view definitions here can often be quite complex, and expensive to compute, so materialization as in a CAM can be quite valuable.

Role-based access control has received considerable attention as a way of unifying mandatory access control and discretionary access control (see, e.g., [Sandhu et al. 1996; Osborn et al. 2000]). Here, permissions are associated with roles created for various job functions in an organization, and one can specify which roles can access which data items. Users are assigned roles based on their responsibilities and can be easily reassigned roles.

9.2 Access Control for XML Documents

There has been considerable recent work on access control for XML documents. Much of this work (see, e.g., [Bertino et al. 1999; 2000b; Bertino et al. 2000a; Bertino et al. 2001; Damiani et al. 2000a; 2000b; Kudo and Hada 2000; Ilioudis et al. 2001]) has studied models for the specification of XML access control policies, typically using discretionary or role-based access control. This body of work focuses on issues

such as granularity of access (e.g., DTD, document, element), propagation options (e.g., local, inherited), and conflict resolution (e.g., most specific, mandatory).

In particular, Bertino et al. [1999] proposed an XML-based formalism for specifying subject credentials and security policies; since these are themselves XML documents, access to the policy can be controlled using the same mechanism. The approach of Damiani et al. [2000a; 2000b] is an extension of their earlier work on object-oriented databases, that defined positive and negative access control, and propagation of access control; their semantics of access control to a user is a particular view of the document(s) as determined by the relevant access control policy. The work in [Bertino et al. 2000a; Bertino et al. 2001] is similar, and is implemented in the Author-X prototype. Kudo and Hada [2000] proposed a provisional access control model for XML documents, where access control decisions on users' actions can be associated with provisional conditions (e.g., signing a statement or logging); an XML access control policy language (XACL) was designed based on this authorization model that integrates a variety of security features. Since CAMs are agnostic to the access control policy, they can be used in conjunction with any of these proposed models.

Mechanisms for the enforcement of these XML access control policies have been studied for the cases of document access (e.g., [Damiani et al. 2000b; Tan et al. 2001]), document browsing and authoring (e.g., [Bertino et al. 2000b; Bertino et al. 2000a]), and querying (e.g., [Cho et al. 2002]). In particular, Damiani et al. [2000b] provide an elegant algorithm for computing the document view accessible to a user using tree labeling. Tan et al. [Tan et al. 2001] designed and implemented an access control system for XML documents, based on the access control model proposed in [Damiani et al. 2000b], where XML documents are stored as relational tables in a relational DBMS. Cho et al. [Cho et al. 2002] focus on secure query evaluation, based on the multi-level security model of mandatory access control, and develop query rewriting and optimization techniques for this purpose. Our work is complementary to this body of work, since none of these papers focus on speeding up the determination of accessibility of individual XML data items.

Finally, the concept of compressed accessibility maps was first introduced in a preliminary version of this paper [Yu et al. 2002]. This paper elaborates on the major theoretical results on optimal CAM construction and its lookup algorithms. In addition, this paper introduces and develops the issue of incremental maintenance of the CAM. We prove that, though optimality cannot be preserved incrementally under data item updates, we can incrementally maintain a near-optimal CAM very efficiently.

9.3 Additional Uses of the CAM

Access control has been central to LDAP implementations (e.g., see [acp 1998]). At any node in the LDAP tree, one can specify a rule that defines permissions or denials for a set of objects in its subtree by a set of users. The set of objects could be specified declaratively, by means of a query. This can necessitate the execution of accessibility queries as a part of the evaluation of the result for a user LDAP query. Our proposal obviates the need for this by materializing this information in the form of a CAM.

More recently, there has been interest in managing access control across multiple

stores (see, e.g., [Candan et al. 1996; Jajodia et al. 1997]). For instance, Jajodia et al. [1997] propose a datalog-like specification language, and show that many of the known meta-policies can be easily encoded in their language, with the intention that different meta-policies can be enforced for different data stores. Once more, materialization of the result of executing their specifications would be of value, and a CAM can be used to store this materialization in compressed form.

10. CONCLUSION

Transacting business over the Internet using XML is becoming more and more of a reality as we move towards a world of inter-networked data, with applications requiring access to data on the net. In this setting, there is a need for much more sophisticated types of access control than is permitted by simple firewalls. We envision that the policies (rules) that define access control to such networked data will interact in complex ways warranting mechanisms for efficient determination of whether or not a user has access to a given data item, given an access type.

The technical contributions of this paper include the design of a linear-time algorithm for finding an optimal CAM (for a given user and access type) for XML databases. With the aid of real-life and synthetic data for multiple users, we demonstrated the substantial savings a CAM can effect on the storage requirements for efficient access control policy enforcement. Last but not the least, we showed that simple updates (additions/deletions) to a database can dramatically change the composition of an optimal CAM. However, with a small price (of a CAM 1 worse than the optimal), we can maintain near-optimality, very efficiently.

An interesting open problem is to take advantage of the commonalities between the access rights of multiple users with respect to parts of the data to optimize the overall space consumed by the CAMs, while still guaranteeing fast lookup time.

We believe that accessibility management should become an integral part of query processing and optimization, especially for XML data on the Web. The compressed accessibility map is an important step towards realizing this vision.

REFERENCES

1998. Access control programmer's guide. Available from http://developer.netscape.com:80/docs/manuals/enterprise/accessapi/contents.htm.

Bertino, E., Braun, M., Castano, S., Ferrari, E., and Mesiti, M. 2000. AuthorX: A Java-based system for XML data protection. In *Proc. of the 14th Annual IFIP WG 11.3 Working Conference on Database Security*. Schoorl, The Netherlands.

Bertino, E., Castano, S., and Ferrari, E. 2001. Securing XML documents with Author-X. *IEEE Internet Computing 5,* 3, 21–31.

Bertino, E., Castano, S., Ferrari, E., and Mesiti, M. 1999. Controlled access and dissemination of XML documents. In *Workshop on Web Information and Data Management*. 22–27.

Bertino, E., Castano, S., Ferrari, E., and Mesiti, M. 2000. Specifying and enforcing access control policies for XML document sources. *World Wide Web Journal (Baltzer Publ.) 3,* 3.

Bertino, E., Samarati, P., and Jajodia, S. 1997. An extended authorization model for relational databases. *IEEE Transactions on Knowledge and Data Engineering 9,* 1, 85–101.

Candan, K. S., Jajodia, S., and Subrahmanian, V. S. 1996. Secure mediated databases. In *Proceedings of the IEEE International Conference on Data Engineering*.

Cho, S., Amer-Yahia, S., Lakshmanan, L. V. S., and Srivastava, D. 2002. Optimizing the secure evaluation of twig queries. In *Proceedings of the International Conference on Very Large Databases*.

Damiani, E., di Vimercati, S. D. C., Paraboschi, S., and Samarati, P. 2000b. Design and implementation of an access control processor for XML documents. *Computer Networks 33,* 1–6, 59–75. Also in WWW9.

Damiani, E., di Vimercati, S. D. C., Paraboschi, S., and Samarati, P. 2000a. Securing XML documents. In *Proceedings of the International Conference on Extending Database Technology.*

Diaz, A. L. and Lovell, D. 1999. XML generator. Available from http://www.alphaworks.ibm.com/tech/xmlgenerator.

Fagin, R. 1978. On an authorization mechanism. *ACM Trans. Database Syst. 3,* 3, 310–319.

Fernandez, M. F., Malhotra, A., Marsh, J., Nagy, M., and Walsh, N. 2002. XQuery 1.0 and XPath 2.0 data model. W3C Working Draft. Available from http://www.w3.org/TR/query-datamodel/.

Ferragina, P. and Grossi, R. 1999. The string B-tree: A new data structure for string search in external memory and its applications. *Journal of the ACM 46,* 2 (Mar.), 236–280.

Gal-Oz, N., Gudes, E., and Fernandez, E. B. 1993. A model of methods access authorization in object-oriented databases. In *Proceedings of the International Conference on Very Large Databases.* 52–61.

Griffiths, P. P. and Wade, B. W. 1976. An authorization mechanism for a relational database system. *ACM Trans. Database Syst. 1,* 3, 242–255.

Ilioudis, C., Pangalos, G., and Vakali, A. 2001. Security model for XML data. In *International Conference on Internet Computing.* 400–406.

Jagadish, H. V., Koudas, N., and Srivastava, D. 2000. On effective multi-dimensional indexing for strings. In *Proceedings of the ACM SIGMOD Conference on Management of Data.*

Jajodia, S., Samarati, P., Subrahmanian, V. S., and Bertino, E. 1997. A unified framework for enforcing multiple access control policies. In *Proceedings of the ACM SIGMOD Conference on Management of Data.*

Jajodia, S. and Sandhu, R. 1991. Toward a multilevel secure relational data model. In *Proceedings of the ACM SIGMOD Conference on Management of Data.*

Kudo, M. and Hada, S. 2000. XML document security based on provisional authorization. In *ACM Conf. Computer and Communications Security.*

Lunt, T. F., Denning, D. E., Schell, R. R., Heckman, M., and Shockley, W. R. 1990. The SeaView security model. *IEEE Trans. Softw. Eng. 16,* 6, 593–607.

Osborn, S. L., Sandhu, R. S., and Munawer, Q. 2000. Configuring role-based access control to enforce mandatory and discretionary access control. *IEEE Trans. on Information Security 3,* 2, 85–106.

Rabitti, F., Bertino, E., Kim, W., and Woelk, D. 1991. A model of authorization for next-generation database systems. *ACM Trans. Database Syst. 16,* 1, 88–131.

Sandhu, R. S. 1993. Lattice-based access control models. *IEEE Computer 26,* 11, 9–19.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. 1996. Role-based access control models. *IEEE Computer 29,* 2, 38–47.

Tan, K.-L., Lee, M.-L., and Wang, Y. 2001. Access control of XML documents in relational database systems. In *International Conference on Internet Computing.* 185–191.

Winslett, M., Smith, K., and Qian, X. 1994. Formal query languages for secure relational databases. *ACM Trans. Database Syst. 19,* 4, 626–662.

Yu, T., Srivastava, D., Lakshmanan, L. V. S., and Jagadish, H. V. 2002. Compressed accessibility map: efficient access control for XML. In *Proceedings of the International Conference on Very Large Databases.* 478–489.

Received July 2003; accepted December 2003