Grid Authorization Graph

Mustafa Kaiiali<sup>a,c,∗</sup>, Rajeev Wankar<sup>a</sup>, C.R. Rao<sup>a</sup>, Arun Agarwal<sup>a</sup>, Rajkumar Buyya<sup>b</sup>

<sup>a</sup> Department of Computer and Information Sciences, University of Hyderabad, Hyderabad, India
<sup>b</sup> Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia
<sup>c</sup> Department of Computer Engineering, Mevlana University, Konya, Turkey

**HIGHLIGHTS**

- A brief overview of access control mechanisms used in grid systems is illustrated.
- The limitations of the Hierarchical Clustering Mechanism (HCM) are highlighted.
- The Grid Authorization Graph (GAG) is introduced to encounter all HCM limitations.
- The GAG Generator Algorithm is illustrated to build GAG decision graph.
- Embedding GAG in GT4 authorization framework is finally discussed.

**ABSTRACT**

The heterogeneous and dynamic nature of a grid environment demands a scalable authorization system. This brings out the need for a fast fine-grained access control mechanism for authorizing grid resources. Existing grid authorization systems adopt inefficient mechanisms for storing resources’ security policies. This leads to a large number of repetitions in checking security rules. One of the efficient mechanisms that handle these repetitions is the Hierarchical Clustering Mechanism (HCM). HCM reduces the redundancy in checking security rules compared to the Brute Force Approach (BFA) as well as the Primitive Clustering Mechanism (PCM). Further enhancement is done to HCM to increase the scalability of the authorization process. However, HCM is not totally free of repetitions and cannot easily describe the OR-based security policies. A novel Grid Authorization Graph (GAG) is proposed to overcome HCM limitations. GAG introduces special types of edges named “Correspondence Edge”/“Discrepancy Edge” which can be used to entirely eliminate the redundancy and handle the cases where a set of security rules are mutually exclusive. Comparative studies are made in a simulated environment using the Grid Authorization Simulator (GAS) developed by the authors. It simulates the authorization process of the existing mechanisms like BFA, PCM, HCM and the proposed novel GAG. It also enables a comparative analysis to be done between these approaches.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Grid computing is concerned with a shared and coordinated use of heterogeneous resources belong to distributed virtual organizations to deliver nontrivial quality of services [1]. In grids, security has a major concern [2]. The heterogeneity, massiveness and dynamism of grid environments complicate and delay the authorization process. This brings out the need for a fast and scalable fine-grained access control mechanism to cater to grid requirements.

Currently, the main focus in the literature is on the way to write the resource’s security policy, either using a standard specification language like SAML/XACML as used in VOMS [3] to provide the interoperability property [4], or it can be specific to a particular authorization system (as in Akenti [5]). Furthermore, they describe the authorization process, either to be centralized (push model [6] as VOMS and CAS [7,8]), or decentralized (pull model [6] as PERMIS [9] and Akenti [10]). Some systems adopt transport level security rather than message level security as the latter involves slow XML manipulations, which make adding security to grid services a performance bottleneck [11].

Current grid authorization systems seldom look at the way in which they store and organize the resources’ security policies in order to work more effectively. There is no well-defined data structure to store and manage the security policies to provide a quick response to the user. There are not so many articles that
have been published so far, and the most representative methods are the Brute Force Approach (BFA) [12] and the Primitive Clustering Mechanism (PCM) [13–15].

Every resource in a grid has its own security policy, which may be identical or quite similar to other security policies of some other resources. This fact motivated us to cluster the resources which have similar security policies in a hierarchical manner based on their shared security rules. The authorization system can built a hierarchical decision tree to find User Authorization Resource Group (UARC). The Hierarchical Clustering Mechanism (HCM) [16–19] was a step in that direction to provide a more fine-grained clustering at multi-levels.

This paper highlights the limitations of HCM and introduces the Grid Authorization Graph (GAG) to overcome these limitations and to further enhance the authorization process by adopting new tools which cannot be adopted in HCM.

Rest of the paper is organized as follows: Section 2 gives a brief description of HCM. Section 3 discusses the proposed GAG and shows how the drawbacks of HCM are addressed. GAG Generator Algorithm is proposed in Section 4. Section 5 explains how GAG components can be embedded in current authorization architecture like GT4. Experiments with results are discussed in Section 6. Section 7 concludes and suggests future work.

2. A brief description of the Hierarchical Clustering Mechanism (HCM)

Consider the following definition:

- Let \( R = \{r_j | j = 1, \ldots, k\} \) be the set of grid resources.
- Let \( \text{SR} = \{sr_j | j = 1, \ldots, l\} \) be the set of security rules.
- Then for each resource \( r_j \in R \) there will be a corresponding security policy \( SP_j \subseteq \text{SR} \).

If a user wants to access resource \( r_j \) then he has to satisfy all the security rules of \( SP_j \). Let us now consider the following example:

A grid environment has 12 resources \( R = \{r_1, r_2, \ldots, r_{12}\} \) and four security rules \( \text{SR} = \{sr_1, sr_2, sr_3, sr_4\} \) where:

- \( sr_1 \) requires the user to be from \( \text{XYZ University} \).
- \( sr_2 \) requires the user to have a \textit{teacher} role.
- \( sr_3 \) requires the user to have a \textit{student} role.
- \( sr_4 \) requires the user to be in \textit{2nd year}.

All the 12 resources are deployed with the following security policies:

- \( r_1, r_2 \) require the user to be from \( \text{XYZ University} \) to be able to access them. So \( SP_1 = SP_2 = \{sr_1\} \).
- \( r_3, r_4 \) require the user to be from \( \text{XYZ University} \) and to have a \textit{teacher} role in order to access them. So \( SP_3 = SP_4 = \{sr_1, sr_2\} \).
- \( r_5, r_6, r_7, r_8, r_9 \) require the user to be a \textit{student} in \( \text{XYZ University} \). So \( SP_5 = SP_6 = SP_7 = SP_8 = SP_9 = \{sr_1, sr_3\} \).
- \( r_{10}, r_{11}, r_{12} \) require the user to be a \textit{2nd year student} in \( \text{XYZ University} \). So \( SP_{10} = SP_{11} = SP_{12} = \{sr_1, sr_3, sr_4\} \).

BFA for the proposed example stores the security policies as shown in Fig. 1. We have 12 security policies each of them consists of a set of security rules, all together need to be checked to find the UARC. Redundancy of BFA is obvious as we have many redundant security policies like \( SP_3, SP_6, SP_7, SP_8 \) and \( SP_9 \).

PCM reduces BFA redundancy by clustering the resources which have identical security policies. Fig. 2 shows how PCM stores the security policies of the proposed example. It is obvious that the number of security policies to be checked is reduced from 12 security policies to only four security policies.

PCM removes the redundancy of checking identical security policies, but it cannot remove the redundancy of checking identical security rules. In other words, it avoids checking identical security policies \( SPs \) more than once; since each security policy \( SP \) is a set of security rules, the security rule \( sr \) level of redundancy is still prevailing in PCM. As an example, \( \text{XYZ University} \) security rule has to be checked four times.

HCM [16] clusters the resources in parent nodes based on their shared security policies, as in PCM. However, it also achieves a hierarchical clustering of these parent nodes based on their shared
security rules to reduce the security rule level of redundancy. Fig. 3 shows HCM representation of the proposed example.

Table 1  
Comparisons of the three mechanisms (total number of resources is 12).

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFA</td>
<td>12</td>
<td>25</td>
<td>2.08</td>
</tr>
<tr>
<td>PCM</td>
<td>4</td>
<td>8</td>
<td>0.67</td>
</tr>
<tr>
<td>HCM</td>
<td>–</td>
<td>4</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Table 2  
Security table example (resources vs. security rules).

<table>
<thead>
<tr>
<th>R&lt;sub&gt;id&lt;/sub&gt;</th>
<th>XYZ</th>
<th>Teacher</th>
<th>Student</th>
<th>2nd year</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&lt;sub&gt;1&lt;/sub&gt;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r&lt;sub&gt;3&lt;/sub&gt;</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r&lt;sub&gt;4&lt;/sub&gt;</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>r&lt;sub&gt;5&lt;/sub&gt;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>r&lt;sub&gt;6&lt;/sub&gt;</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Building HCM decision tree is not a trivial process. An algorithm that properly chooses the root security rule of the tree and its sub-trees is required. For that the Counting Algorithm is proposed in [16]. It is a single-pass, depth-first algorithm developed to build HCM decision tree based on the data of the Security Table (ST).

The Security Table (ST) is a table representation of all resources’ security policies; where security rules are considered as attributes, and resources as objects, with table entries of (i,j)th cell as 1 if the jth security rule is an element of the security policy of the ith resource. Table 2 is the corresponding Security Table for the proposed example. Fig. 3 is the output decision tree when we run the Counting Algorithm on Table 2. Further details on HCM can be found in [16–19].

3. The Grid Authorization Graph (GAG)

In this section, the limitations of HCM are discussed. Then the Grid Authorization Graph (GAG), a decision graph derived from HCM
decision tree by embedding various edges and tools, is introduced to encounter all the issues and limitations of HCM.

3.1. HCM limitations

3.1.1. Describing OR-based security policies

A grid resource may have multiple ways to access it. For example, consider a grid environment of six resources and four security rules represented by the security table shown in Table 3. Resource (r<sub>4</sub>) has two different ways to access it. That is why it has two rows in the security table. A user can access resource (r<sub>4</sub>) if he/she is a student in XYZ University OR a programmer in XYZ Software Company.

HCM decision tree cannot represent r<sub>4</sub> security policy, unless it duplicates r<sub>4</sub> resource node as shown in Fig. 4. In general, if a resource has x ways to access it, then HCM decision tree has to duplicate its resource node x times. This is why HCM cannot easily describe the OR-based security policies.

3.1.2. Redundancy in HCM

Even though HCM reduces the redundancy compared to BFA and PCM, it does not entirely eliminate it. For example, consider a grid environment of 20 resources and five security rules where Table 4 represents the resources’ security policies.

Fig. 5 shows the output decision tree when the Counting Algorithm runs on Table 4; It can be observed that the security rule s<sub>R</sub> has to be checked four times and s<sub>R</sub> has to be checked three times. This shows that HCM does not completely eliminate the redundancy. That leads us to introduce the Grid Authorization Graph (GAG) as discussed in the next sub-section.

3.2. Resolving HCM limitations using GAG

3.2.1. Describing OR-based security policies using GAG

A graph data structure allows a node to be a child of more than one parent node. Thus it can easily describe the OR-based security policies without the need to duplicate any resource’s node. Fig. 6 shows how GAG represents the security policies of Table 3.
3.2.2. Eliminating redundancy of HCM using GAG

Fig. 5 shows an example of redundancy in HCM decision tree. To encounter this issue, GAG introduces a special type of edges named “Correspondence Edge” which can be used to entirely eliminate the redundancy.

Once a security rule, as an example $sr_5$, is checked for the first time, a one level BFS (Breadth-First Search) [20] on its Correspondence Edges is enough to mark all the redundant security rules with the result of the first check. Thus, when the authorization process reaches a redundant node it will find it already marked with the result of the first check and no need to do any further checking.

Consider a user whose credentials satisfy $sr_1$, $sr_2$ and $sr_5$ security rules. Fig. 8 shows the decision graph parsed for this particular user. First, $sr_1$ is checked. As the user satisfies $sr_1$, the system adds resources $r_1$ and $r_2$ to the UARG then it proceeds to check $sr_2$. As the user satisfies $sr_2$, the system adds resources $r_3$, $r_4$, $r_5$, and $r_6$ to the UARG then it proceeds to check $sr_3$. As the user does not satisfy $sr_3$, the whole $sr_3$ sub-tree is marked as “unauthorized” and then the system proceeds to $sr_5$. As the user satisfies $sr_5$, the system adds resources $r_{17}$ and $r_{18}$ to the UARG then it makes a BFS on the Correspondence Edges of $sr_5$ node to mark all redundant $sr_5$ nodes as “authorized” and then all child resources of these redundant nodes, like $r_9$ and $r_{10}$, are added to the UARG. Then the system proceeds to $sr_4$ node. As the user does not satisfy $sr_4$ security rule, a BFS is done on the Correspondence Edges to mark all redundant $sr_4$ nodes as “unauthorized”.

There are two rules to be considered while propagating the authorization result through the Correspondence Edges:

- The “unauthorized” marking dominates the “authorized” marking, so the system cannot mark a redundant node as “authorized” when it has been previously marked as “unauthorized”. As an example, when we have propagated the “authorized” decision to $sr_5$ node in the sub-tree of $sr_3$ node, the whole $sr_5$ sub-tree was already marked as “unauthorized”. So we cannot mark this $sr_5$ node as “authorized”.

- If the parent security rule of a redundant node is not yet checked then we cannot add its child resources to the UARG when we propagate the “authorized” decision until its parent node is checked. As an example, suppose the decision graph is parsed for a user whose credentials satisfy $sr_4$ security rule and does not satisfy $sr_3$. If $sr_4$ security rule is checked before $sr_3$, then while propagating the “authorized” decision to the redundant $sr_3$ node in $sr_3$ sub-tree, we cannot add its child resource $r_{14}$ to the UARG because its parent security rule ($sr_4$) is not yet checked.

#### Table 4

<table>
<thead>
<tr>
<th>$R_id$</th>
<th>$sr_1$</th>
<th>$sr_2$</th>
<th>$sr_3$</th>
<th>$sr_4$</th>
<th>$sr_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_3$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_4$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_5$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_6$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_7$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$r_8$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$r_9$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$r_{10}$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$r_{11}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_{12}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_{13}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_{14}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_{15}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$r_{16}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$r_{17}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$r_{18}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$r_{19}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$r_{20}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
It is useful to mention that, apart from eliminating redundancy, Correspondence Edges provide more functionality listed below:

- Consistency in checking security rules, and
- Can also be used for compatible security rules; that is when different security rules always share the same authorization decision. Compatible security rules have to be defined by the administrator and cannot be discovered automatically.

### 3.2.3 Handling mutually exclusive security rules

Another type of edges which reflects different meaning of dependency can be introduced to handle the case where a set of security rules are mutually exclusive. This type of edges is named as "Discrepancy Edge". Fig. 9 represents the Discrepancy Edge with a black dotted line. It is an edge drawn between each mutually exclusive security rules. It can be read as the following:

“If sr3 is satisfied then sr4 and sr5 cannot be satisfied”.

So it is evident not to check sr4 and sr5 if sr3 is satisfied as it is already known to the system that sr4 and sr5 security rules are mutually exclusive to sr3 and they cannot be satisfied all together. Therefore, with the help of the Discrepancy Edges, once sr3 security rule is checked and satisfied, a one level BFS on the Discrepancy Edges is enough to mark all the set of mutually exclusive security rules sr4 and sr5 as "unauthorized".

Consider a user whose credentials satisfy security rules \{sr1, sr2, sr3\}. Fig. 10 shows the decision graph (shown earlier in Fig. 9) parsed for this particular user. First, sr1 is checked. As the user satisfies sr1, the system adds resources r1 and r2 to the UARG then it proceeds to check sr2. As the user satisfies sr2, the system adds resources r3, r4, r5, and r6 to the UARG then it proceeds to check sr3. As the user satisfies sr3, the system adds resources r11, r12 and r13 to the UARG then it makes a BFS on the Discrepancy Edges of sr3 node to mark all the set of mutually exclusive security rules sr4 and sr5 as “unauthorized”. Correspondence Edges are then used to mark all the redundant sr4 and sr5 nodes as “unauthorized”.

In general, each type of the Dependency Edges (Correspondence/Discrepancy) can have seven forms depicted in Fig. 11. The “One Way” form propagates the result in one direction only, while the “Two Ways” form propagates in two directions. The “Positive” form propagates the “authorized” result only while the “Negative” form propagates the “unauthorized” result only.
4. GAG Generator Algorithm

Dependency Edges can be added manually by the administrator as per system requirements. However, it is infeasible to add all Correspondence Edges between redundant nodes manually. An algorithm which automatically tracks the redundant nodes and draws the Correspondence Edges between them is required.

The Counting Algorithm [16], used earlier to build HCM decision tree is upgraded in this section to build GAG decision graph by adding the Correspondence Edges automatically between redundant nodes. It is named as “GAG Generator Algorithm”. It uses the Security Rules Vector (SRV) to avoid the need to draw a clique between redundant nodes when we are not sure which redundant node is going to be checked first.

The output of the GAG Generator Algorithm when it runs on the security table shown in Table 4 is depicted in Fig. 12. During the authorization process, when the security rule of a particular node is checked, the result is propagated through the undirected edge of that node to its correspondent cell in the SRV. Then the result is further propagated through the correspondent SRV cell to all redundant nodes via one level BFS. Table 5 describes the computational complexity of the GAG Generator Algorithm.

After introducing GAG with its powerful tools, we can notice that: “HCM decision tree is still at the core of GAG”. Figs. 7 and 9 show an example of that. This means all the caching mechanisms, which were designed to work in HCM decision tree like the Temporal Caching Mechanism (TCM) [17], and the Hamming Distance Caching Mechanism (HDCM) [17], are still valid to work in GAG. Thus TCM and HDCM modules of HCM can be embedded directly in GAG Search Engine described in Section 5. Moreover,
all the analysis that have been done to prove the stability of HCM against the dynamic changes in the grid environment [17] is also valid for GAG.

**GAG Generator Algorithm:**

**Inputs:** Resources’ Security Table

**Outputs:** Grid Authorization Graph (GAG)

**Variables:**
1. **SRV**: A vector of all security rules (Fig. 12).
2. Each node \( N \) in the graph is a structure of 3 fields:
   - the security rule \( sr \),
   - an interim security table \( ST \) and
   - an undirected correspondent edge from the node to the representative \( sr \) cell in \( SRV \).

**Begin:**

**Step 1:** (Initialization)
- Initialize the decision tree by a root node with NULL security rule (\( sr \)).
- Build the security table which represents the entire security policies of the system. Assign it as the security table property (\( ST \)) of the root node.
- Assign NULL to the root node correspondent edge.
- Execute Step 2 for the root node.

**Step 2:** (Processing of one node \( N \))

**Step 2.1:** (Adding \( N \)'s Resources)
- Add each resource, whose correspondent row in \( N \)'s \( ST \) has ‘0’ cells, as a child resource to \( N \).
- Choose the security rule \( sr \) with the highest Count.
- Divide \( ST \) into two tables excluding the \( j \)th column as the following:
  - The first table \( (T_1) \) contains the rows of the resources which demand \( sr \) (each row whose \( j \)th cell = 1).
  - The second table \( (T_2) \) contains the rows of the resources which do not demand \( sr \) (each row whose \( j \)th cell = 0).

**Step 2.2:** (Processing of \( N \)'s security table \( ST \))
- Sum the cells of each column of \( N \)'s \( ST \) and refer it as Count.
- Execute Step 2 for the root node.

**Step 3:** (Recurring)
- Repeat step 2 for each child node until a node with empty security table is reached.

**Step 4:** (Pruning)
- Prune the graph at nodes labelled NULL.
- Erase all interim security tables (\( STs \)) to free space.

**End.**

**Table 5**
The average computational complexity of the GAG Generator Algorithm. (\( M \) is the number of resources and \( N \) is the number of security rules.)

<table>
<thead>
<tr>
<th>Step</th>
<th>Complexity</th>
<th>Repeated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>( O(M \times N) )</td>
<td>(1) time</td>
</tr>
<tr>
<td>Step 2</td>
<td>( O \left( \sum_{i=0}^{N-1} 2^i \times \text{Step 2} \right) )</td>
<td>( N + 1 ) times</td>
</tr>
<tr>
<td>Step 3</td>
<td>( O(1) ); Simple condition check.</td>
<td>(1) time</td>
</tr>
<tr>
<td>Step 4</td>
<td>( O(N) ); Maximum NULL nodes is ( N ).</td>
<td></td>
</tr>
</tbody>
</table>

**Total algorithm complexity** \( O(M \times N^2) \); usually \( N \ll M \).

5. Embedding GAG in GT4 authorization framework

GT4 [21] authorization framework [22] was constructed based on the OASIS XACML and SAML standards [23]. It contains the PEP (Policy Enforcement Point) [24], the PDP (Policy Decision Point) [24], thePIP (Policy Information Point) [24] and the PAP (Policy Administration Point) [24]. To make the framework compatible with GAG, five more subcomponents were added to the architecture as shown in Fig. 13. These subcomponents are:
• **RAP (Request Analyzer & Processor)** and GAG Search Engine in the PDP.

• XML Parser, GAG Generator Engine and GAG Database.

The resource’s security policy is submitted by the stakeholder to the PAP through SAML or XACML specification language. Thus an XML Parser is required to parse the security policies’ files, pick up the security rules and provide a simplified input to GAG Generator Engine in the form of Security Table (like Table 2). Typically there are two types of XML parsers, SAX [25] and DOM [26] parsers. DOM Parser is slow and consumes a lot of memory when it loads an XML document that contains a lot of data. SAX is faster than DOM and uses less memory. Thus using SAX parser is strongly recommended in a dynamic and huge environment such as the grid.

GAG Generator Engine is responsible to build the proposed Grid Authorization Graph (GAG) out of the security table provided by the XML Parser. Practically, it is a direct implementation of the GAG Generator Algorithm whose pseudo code is shown in Section 4. Following this, it maintains the output decision graph in GAG Database to be used by GAG Search Engine.

When a user raises an access request, the PEP intercepts the request and propagates it to the PDP. The request is kept in a queue in the PDP. The RAP is a simple Action Listener which listens on the PDP queue. Once a request is enrolled into the queue, RAP picks up the request, fetches the authorization attributes [27] of the correspondent subject (user) from the PIP then it fires an authorization process in the GAG Search Engine to find the UARG.

GAG Search Engine is responsible to parse the decision graph for the incoming requests to find the UARG (Fig. 10). Considering the large number of users and resources which exist in the grid, GAG Search Engine may cause a bottleneck to the authorization system as a centralized process to serve all the incoming authorization requests. This can be solved either by replicating the decision graph into several authorization servers to share the authorization load or by enhancing the search engine itself to serve multiple authorization requests concurrently. However, this is not a special issue of GAG. It is inherited from HCM and has already been addressed by introducing the Concurrent HCM [19]. As HCM decision tree is nestled at the core of GAG, implementing Concurrent GAG Search Engine will be quite similar to implementing Concurrent HCM.

Finally, GAG Search Engine returns the UARG back to RAP based on which RAP will make the access decision to the targeted resources. The access decision is sent back to the PEP. The PEP fulfills the obligations and either permits or denies the access request according to the decision of the PDP.

### 6. Experiments and results

For a grid environment of 200 resources and 15 security rules, 100 different authorization processes have been initiated. For each authorization process, the posterior analysis of HCM and GAG has been done and depicted in Table 6 and Fig. 14 (X axis is for the authorization process number (Experiment No) and Y axis is for the authorization complexity (No of checked security rules)).

Looking at the most important performance metrics, AVG and MAX shown in Table 6, we can realize that GAG outperforms HCM. MAX number of checked security rules in GAG equals the total number of security rules existing in the system because GAG’s redundancy is ZERO. While in case of HCM, due to the redundant nodes in the decision tree, MAX number of checked security rules was quite large as compared to GAG.

It is also important to notice that while GAG entirely eliminates the redundancy in checking security rules, it also adds extra complexity to the authorization process when it does Breadth First Search (BFS) on the Dependency Edges. However, the cost of the BFSs operations compared to the cost of checking the redundant security rules is negligible. As checking a security rule requires

---

**Table 6**

Experiments and results: (Unit is the number of checked security rules).

<table>
<thead>
<tr>
<th></th>
<th>AVG</th>
<th>Standard deviation</th>
<th>Range [MIN, MAX]</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCM</td>
<td>51</td>
<td>41.8616</td>
<td>[6, 207]</td>
</tr>
<tr>
<td>GAG</td>
<td>13</td>
<td>2.4310</td>
<td>[6, 15]</td>
</tr>
</tbody>
</table>

---

**Fig. 13.** GAG enabled authorization framework (shaded components are our contributions).
checking of user credentials (attributes assertions [28] issued by the Attributes Authorities [29]), and this further requires PKI [30] operations, which are known to be expensive processes [31].

NOTE: All experiments are done on the Grid Authorization Simulator (GAS). GAS is a C# based application developed in Grid Computing Laboratory, University of Hyderabad, India. It is used to simulate the authorization process of existing mechanisms like BFA, PCM, HCM as well as our proposed GAG mechanism.

7. Conclusion and future scope

In this paper, a novel grid authorization enhancement is proposed by introducing the Grid Authorization Graph. While HCM reduces the redundancy in checking security rules compared to BFA and PCM mechanisms, GAG eliminates it completely.

As HCM is still at the core of GAG, TCM and HDCM caching mechanisms are still valid for work in GAG and all the analysis, which have been done to prove the stability of HCM against the dynamic changes in the grid are also valid for GAG. GAG introduces special types of edges named Correspondence Edge/Discrepancy Edge which are used to completely eliminate the redundancy and handle the cases where a set of security rules are mutually exclusive, to speed up the authorization process.

This paper also shows how GAG can be embedded in the GT4 authorization framework. Thus, GAG is an efficient and superior access control mechanism which can be integrated in the present popular grid authorizing systems like VOMS, Akenti, PERMIS, etc. The real impact on the performance can be observed if GAG is used in a medium/large environments.

GAG provides the UARG on which a scheduling algorithm has to run later to coordinate job execution among the selected resources. As GAG covers the entire grid resources during the authorization process, one can think of utilizing this process to collect initial information about resources’ availability and other important scheduling parameters to the scheduler which may help to speed up the scheduling process.

One of the things which leads us to develop our own simulator is that existing GridSim does not have an authorization module where we can integrate our mechanisms and test them. One of the future works is to implement an authorization module in GridSim. Moreover, a realtime implementation of GAG in real grid systems such as GridBus and Globus can be a follow-up step.

References

Mustafa Kaiiali is working as Assistant Professor in the Department of Computer Engineering, Mevlana University, Konya, Turkey since 2013. He has completed his undergraduate studies (B.E.) at Alagia Loyola University, after his Bachelor degree he had his postgraduate studies (M.Tech degree) at St. Joseph University. He has served as a Visiting Professor at Tsinghua University, Shanghai, China; National University of Singapore, Singapore; Indian Institute of Technology, Kanpur, India; University of Bochum, Germany; and University of Uppsala, Sweden. He has published many Journal and Conference papers and served as the Guest Editor of IJCSA’s Special issue on Grid and Parallel Systems.

C.R. Rao completed his B.Sc. and M.Sc. in Statistics from Andhra University and Osmania University respectively, Ph.D. in Statistics and M.Tech (CS & Engineering) from Osmania University.

He worked as a lecturer in Statistics at Osmania University since 1986, he is working in the School of Mathematical and Computer/Information Sciences, University of Hyderabad. Presently he is a Professor in the Department of Computer and Information Sciences, University of Hyderabad. His current research interests are Simulation & Modeling and Knowledge Discovery. Rao has been a member of the Operation Research Society of Indian, Indian Mathematical Society, International Association of Engineers, Society for development of statistics, Andhra Pradesh Society for Mathematical Sciences, Indian Society for Probability and Statistics, Society for High Energy Materials, International Rough Set Society, Indian Society for Rough sets, International Rough Set Society and also a Fellow of The Institute of Electronics and Telecommunication Engineers and Society for Sciences.

Arun Agarwal completed his B.Tech (Electrical Engineering) in 1979 and Ph.D. (Computer Science) in 1989 both from IIT Delhi. He started his career as a Senior Research Assistant in IIT Delhi in 1979 and then joined University of Hyderabad in 1984, where at present he is a Professor of Department of Computer/Information Sciences.

Beijing, China; San Diego Supercomputing Centre USA; Bioinformatics Institute in Singapore, Queensland, Australia; NECTEC, Thailand; NCISA, University of Illinois, Urbana-Champaign, USA; USM, Penang, Malaysia; RAL, South Korea; IIT, VAST, Hanoi, Vietnam etc.

He is on the Editorial Board of Editor, Journal of Emerging Technologies in Web Intelligence, International Journal of Pattern Recognition (RBCS); and Engineering Letters of International Association of Engineers. He is also a Fellow of Andhra Pradesh Academy of Sciences, Fellow of IETE, Senior Member of IEEE, USA; Expert Member of AICTE to recognize new colleges to start engineering courses; Member of Board of Studies of several Universities. He was Chairman of IEEE Hyderabad Section for the years 2001 and 2002. He also received the IEEE Region 10 Outstanding Volunteer Award in 2009 in recognition of his dedications and contributions.

He has served on the technical program committee of numerous conferences in the area of Pattern Recognition and Artificial Intelligence. He has served as committee chair of a number of these conferences. He is also on the Steering Committee of PRAGMA, Member APGrid PMA. He is a member of GARUDA project, a national initiative on Grid Computing.

His areas of interest are in Computer Vision, Image Processing, Neural Networks and Grid Computing. He has guided 9 Ph.D. Theses and more than 125 postgraduate dissertations and has published about 90 papers. He has several projects and consultancy in hand with several industry/research laboratories.

Mustafa Kaiiali and Rajkumar Buyya have collaborated on several projects over the years. Kaiiali has published extensively in the field of Grid and Cloud Computing, while Buyya has been a driving force behind the development of cloud computing technologies. Their collaboration has resulted in numerous publications and papers in leading journals and conferences.

Software technologies developed under Prof. Buyya’s leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Prof. Buyya has led the establishment and development of key community activities, including serving as a member of the Steering Committee of the International Conference on Scalable Computing and five IEEE/ACM conferences. These contributions and international research leadership of Prof. Buyya are recognized through the award of “2009 IEEE Medal for Excellence in Scalable Computing” from the IEEE Computer Society TCSC, USA; Manjrasoft’s Aneka Cloud technology developed under his leadership has received “2010 Asia Pacific Frost & Sullivan New Product Innovation Award” and “2011 Telstra Innovation Challenge, People’s Choice Award”. For further information on Prof. Buyya, please visit his cybhome: www.buyya.com.