Future Generation Computer Systems 29 (2013) 1909–1918

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Grid Authorization Graph

Mustafa Kaiiali [a,c,*], Rajeev Wankar [a], C.R. Rao [a], Arun Agarwal [a], Rajkumar Buyya [b]

[a] Department of Computer and Information Sciences, University of Hyderabad, Hyderabad, India

[b] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia

[c] 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.

ARTICLE INFO

Article history:
Received 24 July 2012
Received in revised form
21 March 2013
Accepted 6 April 2013
Available online 22 April 2013

Keywords:
Grid computing
Grid authorization
Access control
Hierarchical Clustering Mechanism
Grid Authorization Graph

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.

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

* Corresponding author.
E-mail addresses: mustafa_kaiiali@ieee.org (M. Kaiiali), wankarcs@uohyd.ernet.in (R. Wankar), crrcs@uohyd.ernet.in (C.R. Rao), aruncs@uohyd.ernet.in (A. Agarwal), raj@csse.unimelb.edu.au (R. Buyya).

0167-739X/$ – see front matter
http://dx.doi.org/10.1016/j.future.2013.04.010

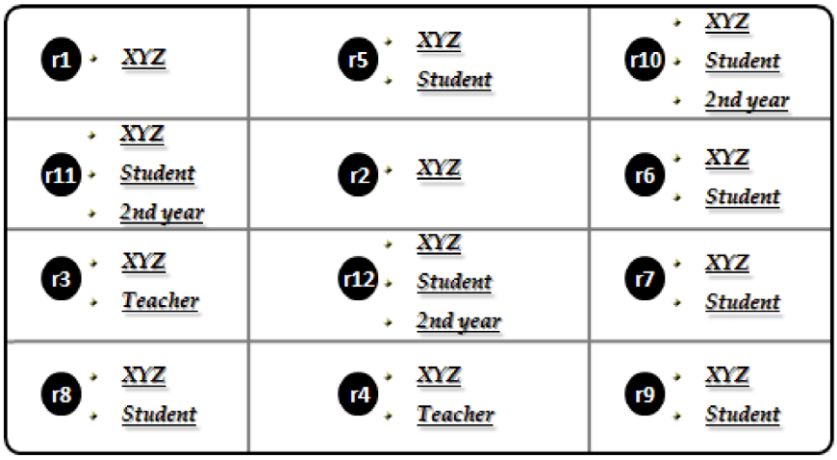

Fig. 1. Example of the Brute Force Approach access control mechanism.

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

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

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

- sr_1 requires the user to be from ***XYZ*** *University*.
- sr_2 requires the user to have a ***teacher*** role.
- sr_3 requires the user to have a ***student*** role.
- sr_4 requires the user to be in ***2nd year***.

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

- r_1, r_2 require the user to be from ***XYZ*** *University* to be able to access them. So $\mathbf{SP_1} = \mathbf{SP_2} = \{sr_1\}$.

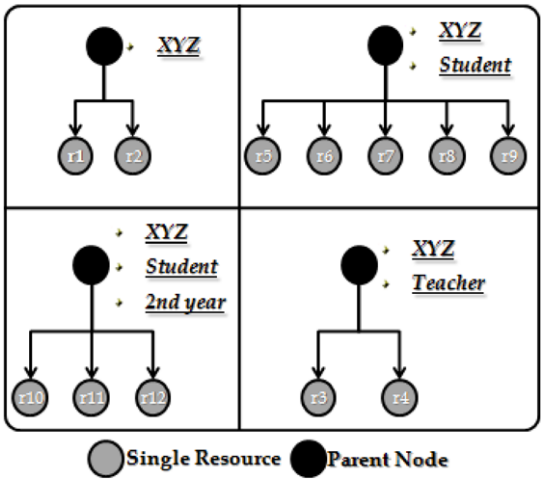

Fig. 2. Example of the Primitive Clustering Mechanism.

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

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 *UARG*. Redundancy of BFA is obvious as we have many redundant security policies like $\mathbf{SP_5}$, $\mathbf{SP_6}$, $\mathbf{SP_7}$, $\mathbf{SP_8}$ and $\mathbf{SP_9}$.

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 **SP**s 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, ***XYZ*** 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

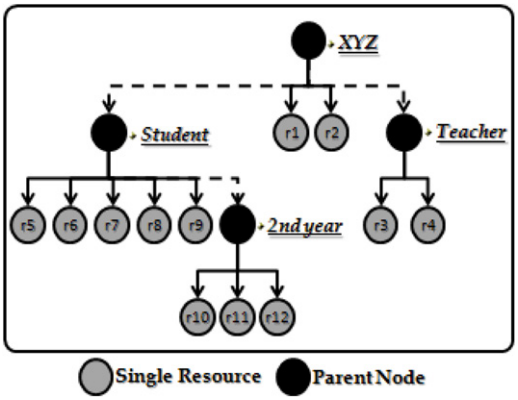

Fig. 3. Example of the Hierarchical Clustering Mechanism.

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

	X	*Y*	*Z*
BFA	12	25	2.08
PCM	4	8	0.67
HCM	–	4	0.33

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

R_{id}	*XYZ*	Teacher	Student	2nd year
r_1	1	0	0	0
r_2	1	0	0	0
r_3	1	1	0	0
r_4	1	1	0	0
r_5	1	0	1	0
r_6	1	0	1	0
r_7	1	0	1	0
r_8	1	0	1	0
r_9	1	0	1	0
r_{10}	1	0	1	1
r_{11}	1	0	1	1
r_{12}	1	0	1	1

security rules to reduce the security rule level of redundancy. Fig. 3 shows HCM representation of the proposed example.

Table 1 presents a comparison between the three mechanisms on the same example discussed earlier. The following parameters are used in the comparison:

- The number of security policies to be checked, say = *x*.
- The number of security rules to be checked, say = *y*.
- The average number of security rules to be checked per single resource = (*y*/total number of resources), say = *z*.

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 *j*th security rule is an element of the security policy of the *i*th 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.

Table 3
Security table example (resources vs. security rules).

R_{id}	*XYZ* Univ.	Student	*XYZ* Soft. Co.	Programmer
r_1	1	0	0	0
r_2	1	0	0	0
r_3	1	1	0	0
r_4	1	1	0	0
r_4	0	0	1	1
r_5	0	0	1	0
r_6	0	0	1	1

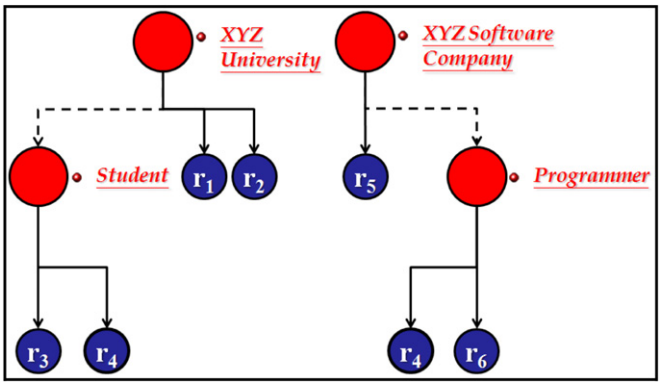

Fig. 4. Describing OR-based security policies in HCM.

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_4) has two different ways to access it. That is why it has two rows in the security table. A user can access resource (r_4) if he/she is a ***student*** in ***XYZ University*** *OR* a ***programmer*** in ***XYZ Software Company***.

HCM *decision tree* cannot represent r_4 security policy, unless it duplicates r_4 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 $\mathbf{sr_4}$ has to be checked four times and $\mathbf{sr_5}$ 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.

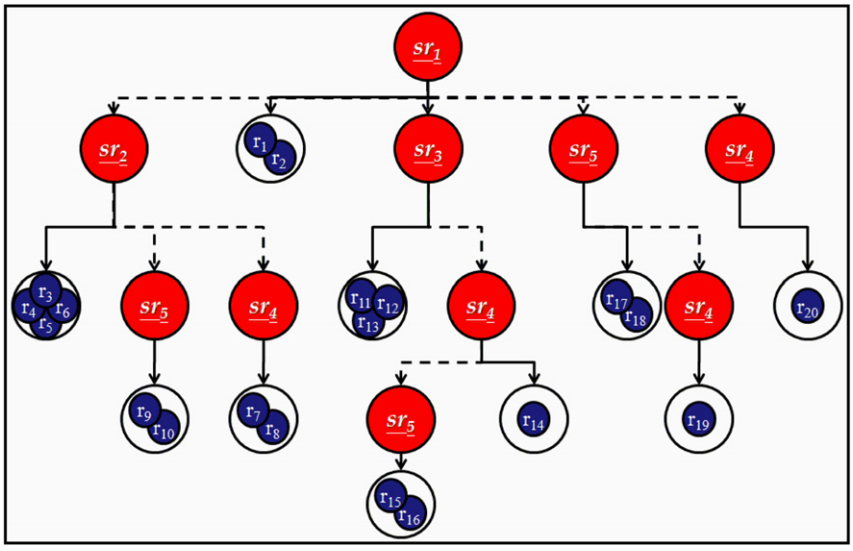

Fig. 5. Redundancy in HCM.

Table 4
Security table example (resources vs. security rules).

R_{id}	sr_1	sr_2	sr_3	sr_4	sr_5
r_1	1	0	0	0	0
r_2	1	0	0	0	0
r_3	1	1	0	0	0
r_4	1	1	0	0	0
r_5	1	1	0	0	0
r_6	1	1	0	0	0
r_7	1	1	0	1	0
r_8	1	1	0	1	0
r_9	1	1	0	0	1
r_{10}	1	1	0	0	1
r_{11}	1	0	1	0	0
r_{12}	1	0	1	0	0
r_{13}	1	0	1	0	0
r_{14}	1	0	1	1	0
r_{15}	1	0	1	1	1
r_{16}	1	0	1	1	1
r_{17}	1	0	0	0	1
r_{18}	1	0	0	0	1
r_{19}	1	0	0	1	1
r_{20}	1	0	0	1	0

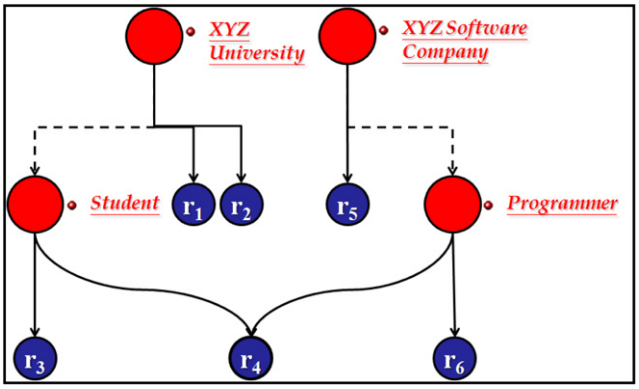

Fig. 6. Describing OR-based security policies in GAG.

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.

Fig. 7 represents the *Correspondence Edge* with a red dotted line. It is an edge drawn between redundant nodes and can be used as the following:

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, then 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_3 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_4 node in sr_3' sub-tree, we cannot add its child resource r_{14} to the *UARG* because its parent security rule (sr_3) is not yet checked.

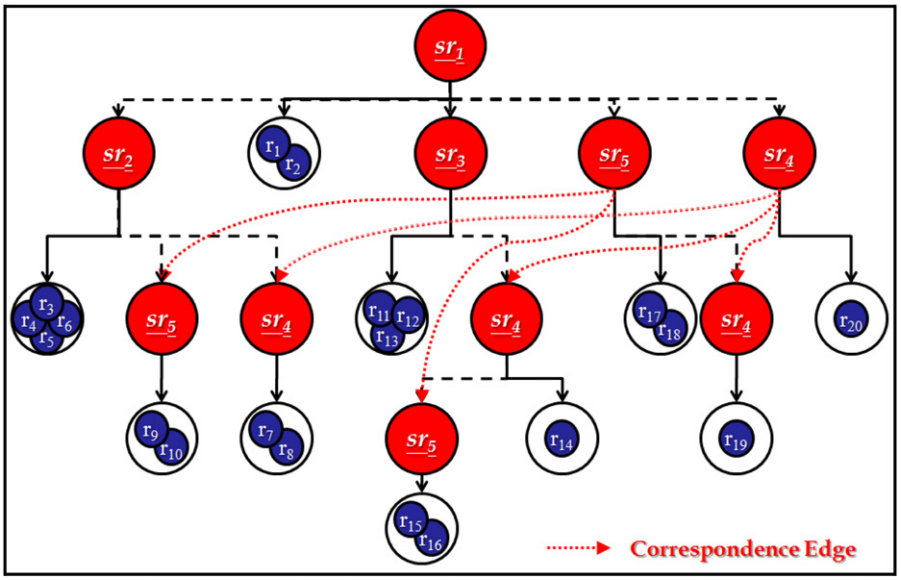

Fig. 7. Eliminating redundancy in GAG.

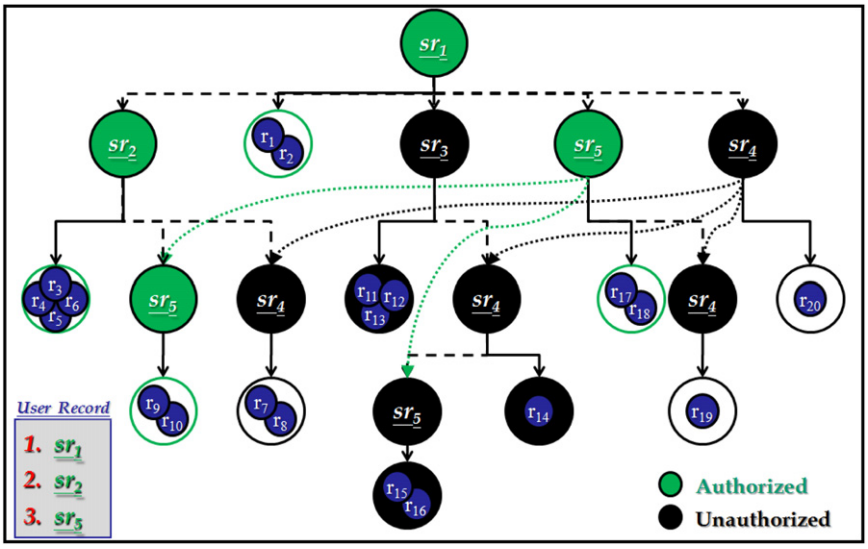

Fig. 8. Example of parsing the *decision graph* with the *Correspondence Edges* in GAG.

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 sr_3 is satisfied then sr_4 and sr_5 cannot be satisfied".

So it is evident not to check sr_4 and sr_5 if sr_3 is satisfied as it is already known to the system that sr_4 and sr_5 security rules are mutually exclusive to sr_3 and they cannot be satisfied all together. Therefore, with the help of the *Discrepancy Edges*, once sr_3 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 sr_4 and sr_5 as "*unauthorized*".

Consider a user whose credentials satisfy security rules $\{sr_1, sr_2, sr_3\}$. Fig. 10 shows the *decision graph* (shown earlier in Fig. 9) 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 satisfies sr_3, the system adds resources r_{11}, r_{12} and r_{13} to the *UARG* then it makes a BFS on the *Discrepancy Edges* of sr_3 node to mark all the set of mutually exclusive security rules sr_4 and sr_5 as "*unauthorized*". *Correspondence Edges* are then used to mark all the redundant sr_4 and sr_5 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.

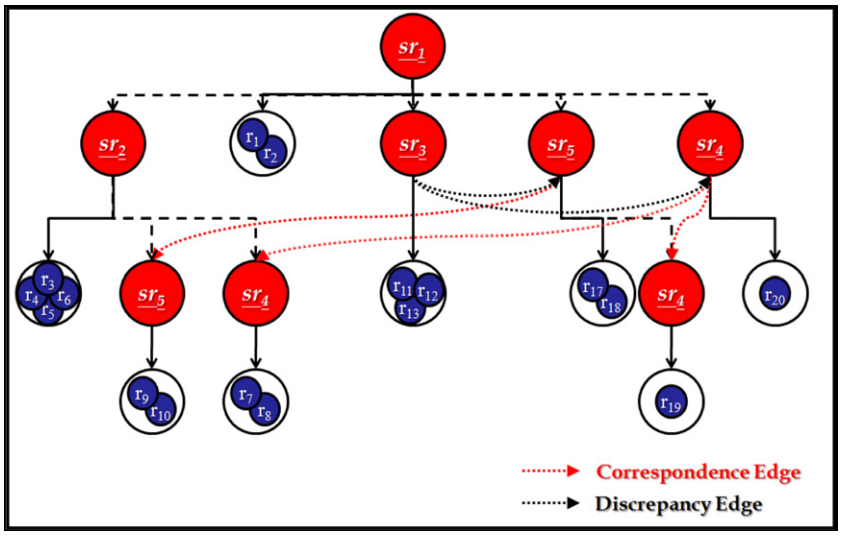

Fig. 9. Handling mutually exclusive security rules in GAG.

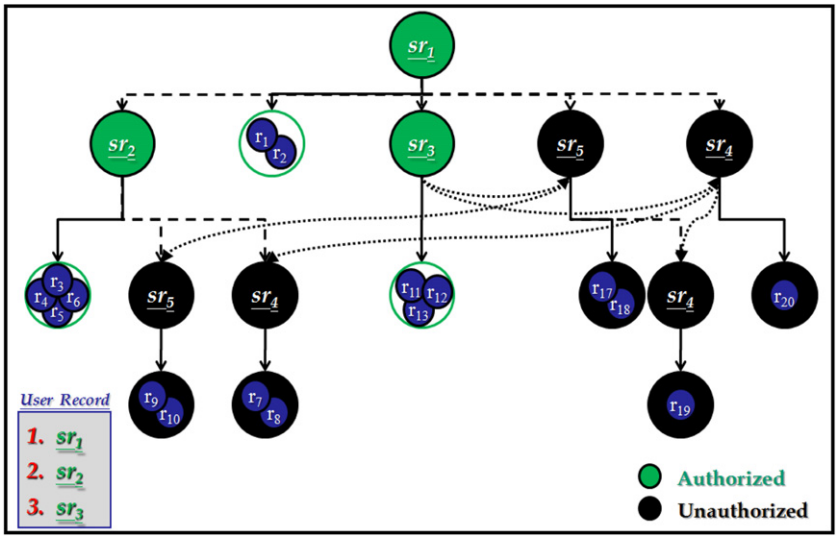

Fig. 10. Example of parsing the *decision graph* with the *Correspondence* and *Discrepancy* Edges.

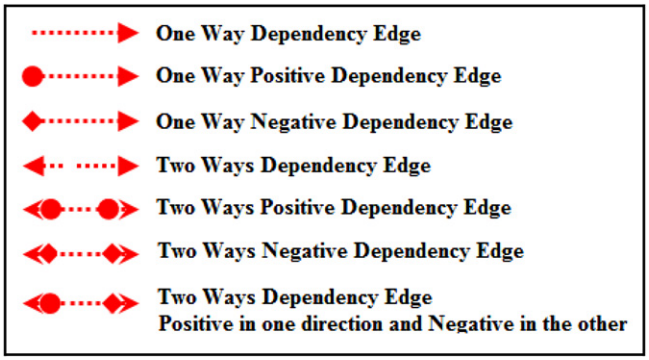

Fig. 11. Different forms of Dependency Edges.

4. GAG Generator Algorithm

Dependency Edges can be added manually by the administrator as per system requirements. However, it is unfeasible 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,

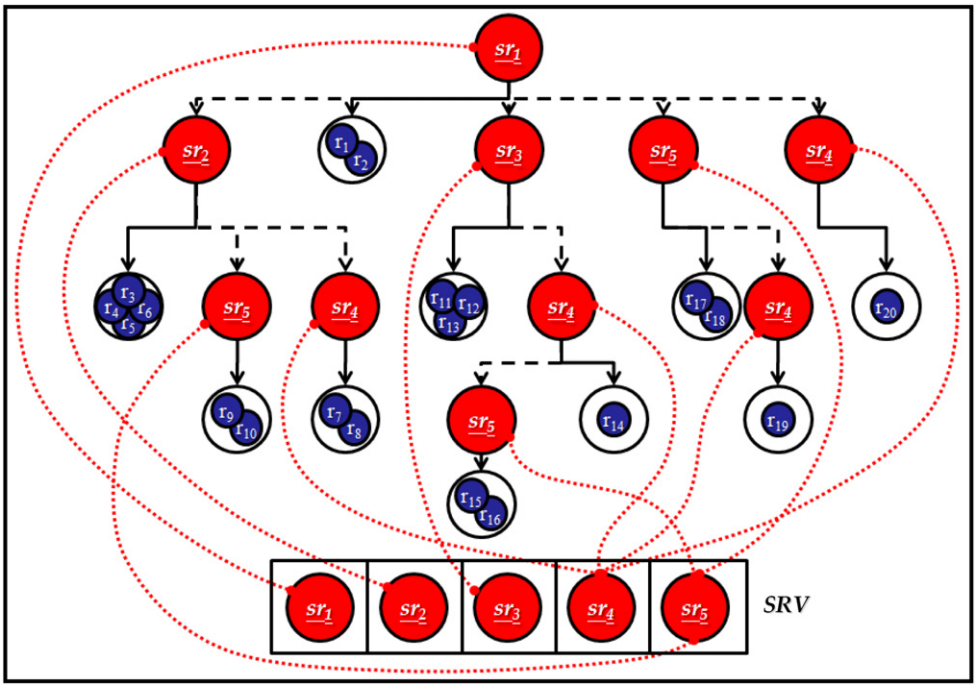

Fig. 12. Example of the GAG Generator Algorithm.

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 *securityrule* (**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**

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**.
- Choose the security rule sr_j with the highest **Count**.
- Divide **ST** into two tables excluding the jth column as the following:
  – The first table ($\mathbf{T_1}$) contains the rows of the resources which demand sr_j (each row whose jth cell = 1).
  – The second table ($\mathbf{T_2}$) contains the rows of the resources which do not demand sr_j (each row whose jth cell = 0).

Step 2.3: (***N Bifurcation***)
- Add a left child node, named "**LCN**", to **N** with sr_j as the *security rule* (**sr**) and $\mathbf{T_1}$ as the *security table* (**ST**). Let the ***correspondent edge*** of **LCN** refer to sr_j's cell in **SRV**.
- Add a right child node, named "**RCN**", to **N** with NULL as the *security rule* (**sr**) and $\mathbf{T_2}$ as the *security table* (**ST**). Let the ***correspondent edge*** of **RCN** refer to NULL.

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* (**ST**s) 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.)

Step	Complexity	Repeated
Step 1	$O(M \times N)$	(1) time
Step 2	$O\left(\frac{M}{2^i} \times (N - i)\right)$: ($i$) is the node's level	$\sum_0^N 2^i \times$ (Step 2)
Step 3	$O(1)$: Simple condition check.	$(N + 1)$ times
Step 4	$O(N)$: Maximum NULL nodes is N.	(1) time
Total algorithm complexity $O\left(M \times N^2\right)$: 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], the *PIP* (*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:

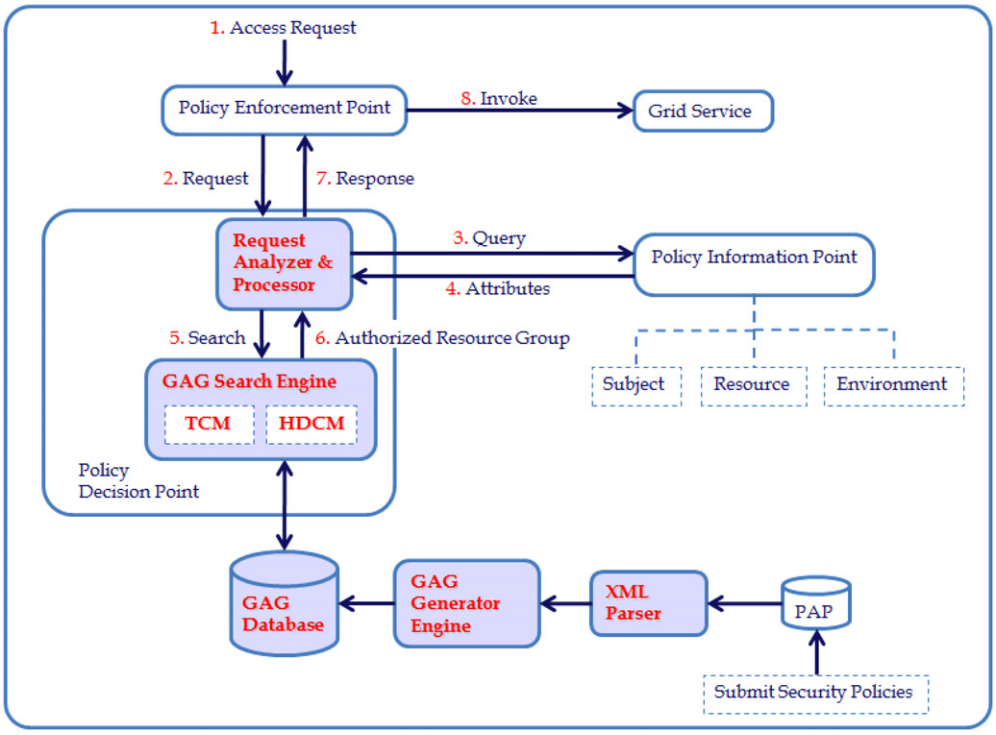

Fig. 13. GAG enabled authorization framework (shaded components are our contributions).

- *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* fulfils the obligations and either permits or denies the access request according to the decision of the *PDP*.

Table 6
Experiments and results: (*Unit* is the number of checked security rules).

	AVG	Standard deviation	Range [MIN, MAX]
HCM	51	41.8616	[6, 207]
GAG	13	2.4310	[6, 15]

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

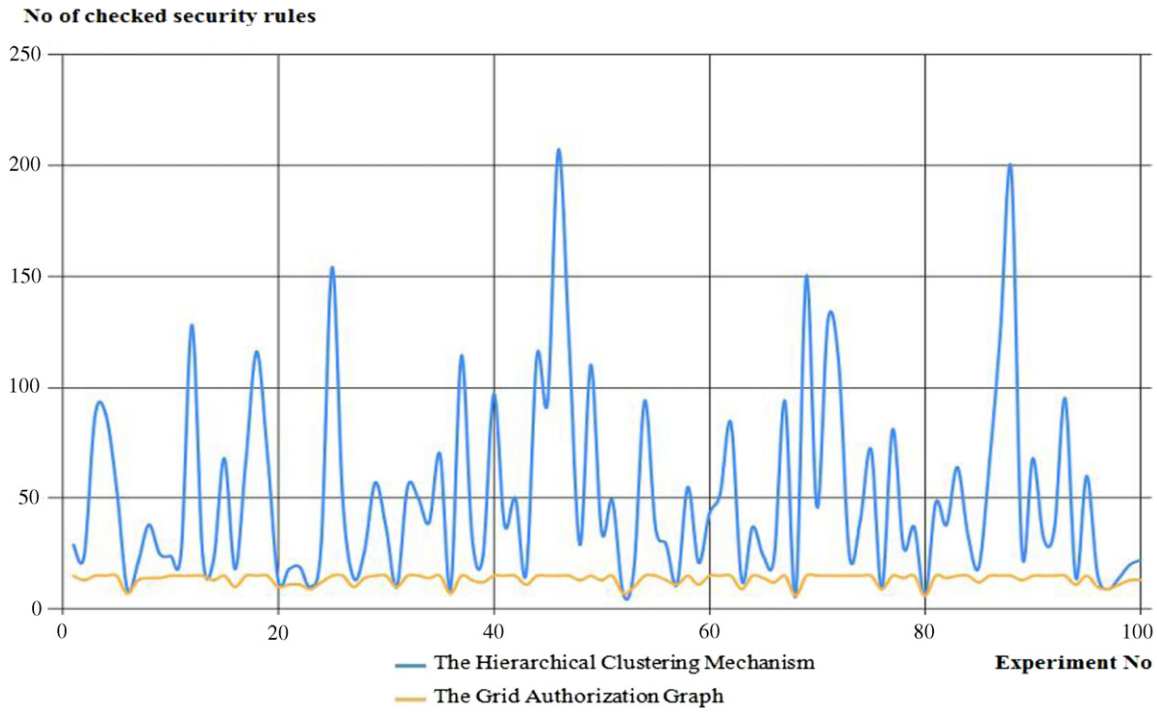

Fig. 14. Experiments and results.

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

[1] I. Foster, What is the grid? A three point checklist, GRID Today (2002).
[2] A. Chakrabarti, A. Damodaran, S. Sengupta, Grid computing security: a taxonomy, IEEE Security & Privacy 6 (1) (2008) 44–51.
[3] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, K. Lörentey, F. Spataro, From gridmap-file to VOMS: managing authorization in a grid environment, Future Generation Computer Systems 21 (4) (2005) 549–558.
[4] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, T. Freeman, A multipolicy authorization framework for grid security, in: IEEE NCA06 Workshop on Adaptive Grid Computing, Cambridge, USA, July 24–26, 2006.
[5] W. Johnston, S. Mudumbai, M. Thompson, Authorization and attribute certificates for widely distributed access control, in: Proceedings of IEEE 7th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises—WETICE'98.
[6] A. Chakrabarti, Grid Computing Security, Springer, 2007.
[7] L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke, A community authorization service for group collaboration, in: IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.
[8] L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke, The community authorization service: status and futures, in: Computing in High Energy Physics, CHEP03, 2003.
[9] D.W. Chadwick, O. Otenko, The PERMIS X.509 role based privilege management infrastructure, in: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies, SACMAT 2002, June 2002.
[10] M. Thompson, A. Essiari, S. Mudumbai, Certificate-based authorization policy in a PKI environment, ACM Transactions on Information and System Security (TISSEC) 6 (4) (2003) 566–588.
[11] S. Shirasuna, A. Slominski, L. Fang, D. Gannon, Performance comparison of security mechanisms for grid services, in: Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing, 8 November 2004, pp. 360–364.
[12] A. Hoheisel, S. Mueller, B. Schnor, Fine-grained security management in a service-oriented grid architecture, in: Proceedings of the Cracow Grid Workshop, Poland, 2006, pp. 433–440.
[13] E. Bertino, P. Mazzoleni, B. Crispo, S. Sivasubramanian, Towards supporting fine-grained access control for Grid resources, in: 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, FTDCS'04, pp. 59–65.
[14] W. Johnston, S. Mudumbai, M. Thompson, Authorization and attribute certificates for widely distributed access control, in: Proceedings of Seventh IEEE International Workshop on Enabling Technologies: Infrastucture for Collaborative Enterprises, WET ICE'98, 1998, pp. 340–345.

[15] M. Kaiiali, R. Wankar, C.R. Rao, A. Agarwal, A rough set based PCM for authorizing grid resources, in: Proceeding of IEEE 10th International Conference on Intelligent Systems Design and Applications, ISDA, Cairo, 29th November 2010, pp. 391–396.
[16] M. Kaiiali, R. Wankar, C.R. Rao, A. Agarwal, Design of a structured fine-grained access control mechanism for authorizing grid resources, in: IEEE 11th International Conference on Computational Science and Engineering, São Paulo, Brazil, 16–18 July 2008, pp. 399–404.
[17] M. Kaiiali, R. Wankar, C.R. Rao, A. Agarwal, Enhancing the hierarchical clustering mechanism of storing resources' security policies in a grid authorization system, in: The 6th International Conference on Distributed Computing and Internet Technology, ICDCIT, in: LNCS, vol. 5966, Springer, 2010, pp. 134–139.
[18] M. Kaiiali, R. Wankar, C.R. Rao, A. Agarwal, New efficient tree-building algorithms for creating HCM decision tree in a grid authorization system, in: The 2nd International Conference on Network Applications Protocols and Services, NETAPPS, Malaysia, 22–23 September 2010, pp. 1–6.
[19] M. Kaiiali, R. Wankar, C.R. Rao, A. Agarwal, Concurrent HCM for authorizing grid resources, in: The 8th International Conference on Distributed Computing and Internet Technology, ICDCIT, in: LNCS, vol. 7154, Springer, 2012, pp. 255–256.
[20] BFS (Breadth-First Search). http://en.wikipedia.org/wiki/Breadth-first_search.
[21] Globus Toolkit 4. http://www.globus.org/toolkit/.
[22] GT4 Authorization Framework. http://www.globus.org/alliance/events/sc06/AuthZ.pdf.
[23] OASIS, eXtensible Access Control Markup Language (XACML), V2.0, January 2005.
[24] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence, AAA authorization framework, IETF, RFC 2904, August 2000.
[25] Y. Pan, Y. Zhang, K. Chiu, Hybrid parallelism for XML SAX parsing, in: IEEE International Conference on Web Services, ICWS08, 23–26 September 2008, pp. 505–512.
[26] T. Takase, K. Tajima, Lazy XML parsing/serialization based on literal and DOM hybrid representation, in: IEEE International Conference on Web Services, ICWS08, 23–26 September 2008, pp. 295–303.
[27] S. Farrell, R. Housley, An Internet attribute certificate profile for authorization, IETF, RFC 3281, April 2002.
[28] M. Lorch, B. Cowles, R. Baker, L. Gommans, P. Madsen, A. McNab, L. Ramakrishnan, K. Sankar, D. Skow, M.R. Thompson, Conceptual grid authorization framework and classification, GGF, 2004. http://www.gridforum.org/documents/GFD.38.pdf.
[29] S. Turner, S. Chokhani, Clearance attribute and authority clearance constraints certificate extension, IETF, RFC 5913, June 2010.
[30] C. Adams, S. Farrell, Internet X.509 public key infrastructure certificate management protocols, IETF, RFC 2510, March 1999.
[31] Developing and maintaining a PKI can be expensive. http://www.tpub.com/content/cg2001/d01277/d012770050.htm.

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 Aleppo University. After his Bachelor degree he had his postgraduate studies (M.Tech degree) and (Ph.D. degree) at the Department of Computer and Information Sciences (DCIS), University of Hyderabad, India. His areas of expertise are: Algorithms, Networking, Information Security, Parallel and Distributed Computing, Grid & Cloud Computing, and Database Systems. His current research work focus is on Grid Security. He has publications in many IEEE and Springer proceedings.

Rajeev Wankar is working as an Associate Professor in the Department of Computer and Information Sciences at University of Hyderabad since July 2004. Before joining this University he was serving as a Lecturer in the Department of Computer Sciences of North Maharashtra University Jalgaon for ten years. He earned Ph.D. in Computer Science from the Department of Computer Science, Devi Ahilya University Indore. In 1998, the German Academic Exchange Service awarded him "Sandwich Model" fellowship. He was working in the Institut für Informatik, Freie Universität, Berlin and had collaboration with Scientists of Konrad Zuse Institut für Informationstechnik (ZIB), a Supercomputing Laboratory in Berlin, for almost two years. Currently he is working in the area of Parallel Computing, especially Parallel Algorithms design using Reconfigurable Bus System, Distributed Shared Memory Computing, Grid Computing and Multi Core Computing. He is actively participated in an International Geo-Grid activity known as GEON with San Diego Supercomputing Centre, University of California, San Diego. He served as a program committee member in many prestigious conferences such as HiPC-07, TEAA, ICDCIT, TENCON etc. He 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 Mathematics 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. Prof. Rao is 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 Institution of Electronics and Telecommunication Engineers and Society for Sciences.

Prof. Rao Guided 5 Ph.Ds, 40 M.Techs, 8 M.Phils. He has nearly 40 Journal and 80 Proceeding Papers. He is Co-author for a book on 'Evaluation of Total Literacy Campaigns'.

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.

Prof. Agarwal was a Visiting Scientist at The Robotics Institute, Carnegie-Mellon University, USA and Research Associate at Sloan School of Management, Massachusetts Institute of Technology, USA. He has also visited, Monash and Melbourne University in Australia; National Center for High Performance Computing, Hsinchu, Taiwan; Chinese Academy of Sciences, Beijing, China; San Diego Supercomputing Centre USA; BioInformatics Institute in Singapore, Queensland, Australia; NECTEC, Thailand; NCSA, University of Illinois, Urbana-Champaign, USA; USM, Penang, Malaysia; KISTI, South Korea; IOIT, 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 Akademi 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 chairs 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. Thesis and more than 125 postgraduate dissertation and has published about 90 papers. He has several projects and consultancy in hand with several industry/research laboratories.

Rajkumar Buyya is Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored over 400 publications and four text books. He also edited several books including "Cloud Computing: Principles and Paradigms" (Wiley Press, USA, Feb 2011). He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 62 and 18 700 + citations).

Software technologies for Grid and Cloud computing 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 foundation Chair of the IEEE Technical Committee 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 cyberhome: www.buyya.com.