Flexible Authorization with Decentralized Access Control Model for Grid Computing

Xinwen Zhang [1], Qi Li [2], Jean-Pierre Seifert [13], Mingwei Xu [2]
[1]*Samsung Information Systems America, San Jose, CA, USA*
{xinwen.z, j.seifert}@samsung.com
[2]*Department of Computer Science, Tsinghua University, Beijing, China*
{liqi, xmw}@csnet1.cs.tsinghua.edu.cn
[3]*Applied Security Research Group, University of Haifa, Haifa, Israel*

Abstract

With the increasing complexity of dynamic and collaborative computing environments in Grid, security management has become a critical factor. Although several approaches have been proposed, fully decentralized and efficient authorization management is still a challenging problem. We propose an access control scheme based on a group-based RBAC model for Grid computing environments. By separating the administrations of users by VO level policies and permissions by resource or service provider policies, our scheme provides decentralized, autonomous, and fine-grained security management which fits the dynamic environment of Grids, and can support ad-hoc collaborations. We implement a proof-of-concept prototype system by enhancing the access control module in Grid File System (GFS) and specifying different levels of policies with XACML.

1 Introduction

Grid computing has been becoming a general platform for automatic, transparent, and pervasive collaborations between various resource and service providers, which are typically formed as virtual organizations (VOs) [9]. As a fundamental problem, authorization is a critical factor for many applications where sensitive operations need to be granted to only authorized entities (subjects) from different organizations (or domains). Particularly, in a Grid-based application, a resource or service provider wants to specify who can access its shared objects. However, due to heterogeneous and dynamic environment, the provider cannot determine subject identities when define policies, since usually, (1) access requests come from different domains or organizations within a VO and users can join or leave the VO community dynamically, and (2) a domain or organization has its own policies determining who can access resources shared by other domains in the community, and (3) these policies can be changed without notice of other domains. Frequently, an authorization decision may require security policies from multiple resources, typically from resource providers and VO [14]. Therefore, authorizations should ensure the ultimate control of resource owners, and autonomous authorization administration in distributed communities. Obviously, traditional approaches with centralized security administration become infeasible.

Several approaches have been proposed in Grid environment for scalable and efficient authorizations, such as Community Authorization Services (CAS) [20], Virtual Organization Membership Service (VOMS) [3], Akenti [27], PRIMA [17], and PERMIS [7] (see Section 5 for related work). These approaches, to some extend, can support various and flexible access control policies on different Grid infrastructures. However, fully decentralized and autonomous authorization management for general collaborations with Grids is still an open problem. Desired solution should not only be flexible and fine-grained, but also be easy to deploy and manage thus meet multi-lateral security requirements.

Role-based access control (RBAC) and its variants have been proposed and deployed in many systems, which have been proven to be able to simplify security administrations and provide efficient policy management [8, 12, 22]. The essential concept of RBAC is to define roles each of which is a collection of permissions that can be invoked to access protected resources. A user is assigned to a set of roles such that obtains the permissions of the roles. However, traditional RBAC model focuses on single and closed security domain, where users are known before authorization and roles are pre-defined [21, 18]. Thus they cannot be directly used in Grid-like environments.

In this paper we propose a two-layer approach to en-

able scalable and efficient authorization management in dynamic and collaborative Grid environment by leveraging RBAC. In the high (or abstract) layer, we propose a group-based RBAC policy model (called GB-RBAC). Extended from RBAC, GB-RBAC simplifies authorization management by introducing the concept of *group*. Specifically, a group is associated with a set of roles defined by *system-level* administrators. A *group-level* administrator can assign users to these roles while the permissions of these roles are controlled by system-level administrators. This simplifies system-level user-role assignments, which are typically centrally managed by few security officers. In the low (or more concrete) layer, we define two authorization schema by following GB-RBAC policy model in Grid environment with VO-managed and ad-hoc Grid computing, respectively. Specifically, within a virtual organization, a resource or service owner defines roles for VOs, which are the permission interfaces to access shared resources or services from VOs. VO or resource consumer organizations assign *local* users to roles in a group. Through this, our scheme enables fine-grained access control in group level but simplified authorization for original resource or service providers. As one of the benefits, our approach is extended from standard RBAC model [8, 4], which has been developed in many organizations. This makes our solution easy to be deployed based on current implementations.

For summary, the contributions of this paper are threefold.

1. We propose the GB-RBAC, which is based on RBAC96 [22] model and extended with group concept. We discuss how decentralized and autonomous security administrations are implemented with our model.

2. We propose two authorization schema in Grid environment to deploy GB-RBAC. Our schema support not only pre-established VO level policies, but also ad-hoc collaboration policies.

3. We implement a proof-of-concept prototype system based on Grid File System (GFS) [16, 25] to show the feasibility our approach. The performance evaluation shows that our approach introduces acceptable overhead.

The paper is organized as follows. Section 2 introduces GB-RBAC model and its administration. Decentralized authorization schema in Grid are illustrated in Section 3. We describe our implemented system with proposed schema in Section 4. Section 5 and 6 present related work and the conclusion of this paper, respectively.

2 Group-based RBAC Model and Administrations

2.1 GB-RBAC Model

GB-RBAC incorporates the component of groups into traditional RBAC model. As RBAC96 model [22] and its administrative models have been extensively studied in literature, we build our model based on them. Figure 1 shows the components of GB-RBAC. The concepts of users (U), roles (R), role hierarchy (RH), permissions (P), permission-role assignment (PA), and sessions (S) are identical to those in RBAC96. Particularly, a permission is an access action to a protected resource or object. A role is assigned with a set of permissions, and a user is assigned to a set of roles, both by security officers or system administrators. In a session, a user activates a subset of assigned roles and obtains the permissions assigned to these roles. Roles can form a partial order hierarchy such that a senior (or higher level) role can inherit the permissions of its junior (or lower level) roles. By configure permission-role and user-role assignments, many security objectives can be achieved efficiently, such as least of privilege and separation of duty. Constraints can be defined for sessions, role hierarchy, and user-role and permission-role assignments for fine-grain authorization controls, such as, a user cannot activate two conflict roles in a single session for dynamic separation of duty purpose.

Besides these, GB-RBAC includes a set of groups (G). Each group is associated with a set of roles (group-role assignment or GA). A user can be the membership of one or more groups, which is represented as the user-group mapping (UM). In addition, we propose two layers of roles which are referred as system-level roles (SR) and group-level roles (GR). Typically, in a GB-RBAC system, a system-level role associates global permissions which manage system-level resources, while a group-level role associates permissions in a small scope, for example, a particular application with specific resources, or a temporary collaborative task between some users.

Besides the user-role assignment in system scope which is similar to the user-role assignment of ARBAC97 (URA97) [21], there is another type of user-role assignment which happens in group scope. Specifically, as UM associates users with groups and GA associates roles to groups, a group administrator can assign a user in the UM to a role in the GA, which is called group-level user-role assignment (GUA), while the original one is called system-level user-role assignment (SUA). In another word, GUA serves as the mechanism through which a role can be assigned to a user because the user *belongs* to a group and the role is assigned to the group, and then the user holds the permissions to access resources defined with the role.

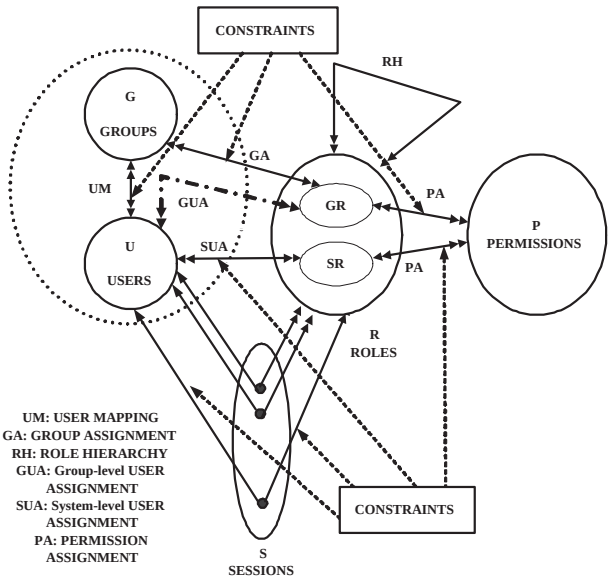

Figure 1. GB-RBAC model

The formal definitions of individual components in GB-RBAC are as follows.

Definition 1 *A GB-RBAC model has the following components:*

- U, P, SR, GR, S, and G *(users, permissions, system-level roles, group-level roles, sessions, and groups, respectively).*
- $R = SR \cup GR$. *For simplicity we assume* $SR \cap GR = \emptyset$ *in this paper.*
- $PA \subseteq P \times R$, *a many-to-many permission to role assignment relation.*
- $UM \subseteq U \times G$, *a many-to-many user to group mapping relation. This relation shows that a user can be mapped into many different groups.*
- $GA \subseteq G \times R$, *a many-to-many group to role assignment relation.*
- $SUA \subseteq U \times SR$, *system-level user-role assignment.*
- $GUA \subseteq U \times GR$, *group-level user-role assignment, and* $(u, r) \in GUA$ *only if* $\exists g, (u, g) \in UM \wedge (g, r) \in GA$.
- $UA = SUA \cup GUA$, *a many-to-many user-role assignment relation.*
- $RH \subseteq R \times R$, *a partial order on R called the role hierarchy or role dominance relation. For any two roles* r_1 *and* r_2, $r_1 \geq r_2$ *means that* r_1 *has partial relation over* r_2.
- $user : S \rightarrow U$, *a function mapping each session* s *to a single user.* $user(s)$ *is constant within* s.
- $permissions : R \rightarrow 2^P$, *a function mapping a role to a set of assigned permissions.*
- $roles : S \rightarrow 2^R$, *a function mapping a session to a set of roles, and* $roles(s) \subseteq \{r | (\exists r' \geq r)[(user(s), r') \in UA]\}$, *which may change within session* s, *and session* s *has the permissions* $\bigcup_{r \in roles(s)} \{p | (\exists r'' \leq r)[(p, r'') \in PA]\}$

A GB-RBAC model can have constraints defined on many aspects shown in Figure 1. Besides constraints on SUA, PA, RH, and sessions, which are similar to those in RBAC96, GB-RBAC can introduce new constraints on UM, GA, and GUA. This paper does not explain detailed specifications of constraints in GB-RBAC.

The procedure to determine the permissions of a user in GB-RBAC is described as follows. When a user logins a system or starts an application, a session is created and a subset of the assigned roles of the user is activated. The set of assigned roles of a user includes the user's directly assigned roles (through SUA), and the roles that are assigned by group-level administrators (through GUA). The user obtains all the permissions assigned to these roles through PA. User can also change the activated roles in a session within his assigned roles. The session can be terminated by the user or by the system, e.g., because of a long idle duration.

In GB-RBAC, a user can be assigned to a group-level role by local (group) administrators if the user belongs to the group, according to group-level authorization policies. In another word, GUA serves as the mechanism through which a role can be assigned to a user because the user belongs to a group and the role is assigned to the group, and then the user holds the permissions to access resources defined with the group-level role.

Note that in general, a user can be assigned to both group-level roles and system-level roles by different levels of administrators, although in a real system s/he may only be assigned to one level of roles.

2.2 Administration of GB-RBAC

The administration of GB-RBAC includes management of all relations of the model, such as UA, PA, and RH. The interesting problem with GB-RBAC, which is different from RBAC96, is that the components in GB-RBAC can be managed by different administrators. Particularly, in our model, permission-role assignment (PR) and group-role assignment (GA) are defined by system administrators

while group-level user-role assignment (GUA) is defined by group administrators. This ensures the separation of administrative privileges. Two-level administrations, referred as system-level and group-level administration, respectively, are proposed to manage the relations defined in GB-RBAC.

For these two administration levels, two types of administrative roles are defined, called system-level administrative roles (SAR) and group-level administrative roles (GAR). These administrative roles can also form role hierarchies, respectively, similar to that of the regular roles in GB-RBAC. For simplicity, we assume that $SAR \cap GAR = \emptyset$. A user of a system administrative role (or simply, a system administrator) can assign a role to a group (through GA), and a user of a group administrative role (or simply, a group administrator) can determine which role the user can be assigned to. In this way, a type of separation of duty in different levels of administration is provided.

A principle of GB-RBAC administration is that permission-role assignment (PA) is determined by system administrators thus controls the final accesses to resources. For user-role assignment, the following components in GB-RBAC have to be managed: GA, UM, SUA, and GUA. Specifically, GA and SUA are in the scope of system administrator, while GUA is in the scope of group administrators. UM can be controlled in both levels, which are determined by organization and application requirements. Typically in Grid environment, each organization has its own policies for enrolling users to different applications, thus UM is determined by group-level administrative model. Note that we do not address the administration of UA since it can be implemented by the administration of SUA and GUA through the Definition 1.

Multi-layer administration provides an autonomy mechanism such that local administrators of a group can assign a member of the group to different roles if some assignment conditions are satisfied, which can be defined as local policies of a group. Administration policies are group and application specific. For example, a VO policy may require that a user can be assigned to a specific role if s/he belongs to a particular group and has been assigned with some particular pre-requisite roles [21, 18, 15]. Other general conditions can be defined in groups for local administrations such as obligations and temporal restrictions, which are not the scope of this paper.

2.3 Benefits of GB-RBAC

With above introduced GB-RBAC and its administration, we summarize the benefits of this authorization mechanism as follows.

1. *Simplified User-role Assignment for System Administrators* In our model, an administrator only needs specify the role range of a group through GA relations. After that, group administrators take charge of the user-role assignment in this local role range. This significantly simplifies the management task by delegating administrative permissions from centralized system-level administrators to decentralized group-level administrators, especially for dynamic and ad hoc collaborative applications.

2. *Flexible Administration for Dynamic User-role Assignment* Group-level administration can flexibly support dynamic user participation in group-based applications, as group-level administrators typically can easily obtain user activities. With purely system-level administration like that in traditional RBAC models, it is tedious to manage user-role assignment in dynamic environments.

3. *Fine-grained User-role Assignment* By enabling GUA, our model supports fine-grained user-role assignment in group level. Typically, a group administrator has more contextual information about local security requirements in the group and the users' skills, thus user-role assignment in this level provides fine-grained and context-aware authorization.

4. *Tunable Group-level Administrations* A system-level administrator can change the role assignment of a group, thus change the roles that a group administrator can assign users to. This greatly provides flexible and controlled group-level administrative permissions.

3 Decentralized Authorization Schema in Grids with GB-RBAC

Based on above defined GB-RBAC model, this section illustrates how it can be used in Grid environment for decentralized authorizations. In a high level view, our scheme enables a resource or service provider to ultimately control the permissions to access its shared objects, and at the same time, give fine-grained user authorizations to virtual organizations. We distinguish two different authorization management schemas: VO managed authorization and ad-hoc authorization. Specifically, in the first scheme, we leverage the centralized management capability of VO for group-level user-role assignment; while in the second scheme, administrators in individual participating domains conduct user-role assignments. In both schema, permissions are assigned to roles by resource providers, which are regarded as system-level administration in GB-RBAC, while user-role assignments are managed by VO or other organizations, which are regarded as group-level administration in GB-RBAC.

As a pre-requisite, trust relations between participating domains and VO are needed in our schema. Without loss of

generality we assume that with internal (e.g., VO-level certificate authority) or external authorities (e.g., trusted third party), each party in our schema can get credentials (e.g, through public key infrastructure) of other parties and build trust relation between them. This mutual trust is the foundation of authentication between participating parties and the assurance that service and security agreements are followed.

3.1 Authorization Scheme in VO-managed Collaborations

Consider a VO (say VO1) among different domains shown in Figure 2. Suppose a domain (say Dom1) shares resource and defines policies to authorize users from different other domains to access the shared resources. Typically, VO level security administrators assign permissions to users from different organizations, e.g., based on user responsibilities or credentials.

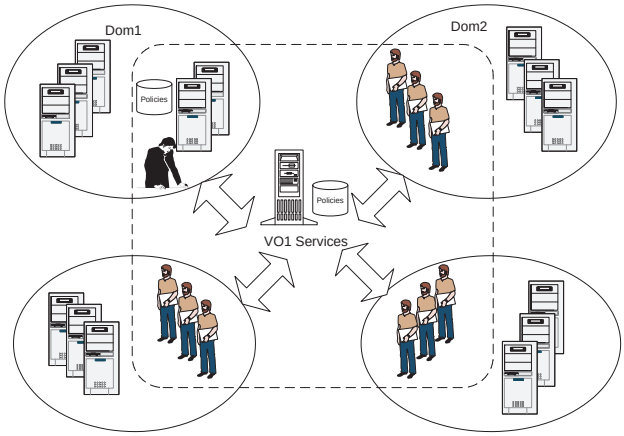

Figure 2. Resource sharing and user authorization in virtual organization

As a resource owner, Dom1 has a set of policies to control accesses to shared resources from VO1 and any other VOs. The goal of these policies is to reserve the final control of Dom1 while provide management flexibility to VO1. The following interactions show the authorization workflow for this purpose.

1. Before sharing resources to VO1, group g_{vo1} is created by Dom1's security administrators. Typically, Dom1 has its own policies to control the permissions to decide shared resources and create groups, which are out of the scope of this paper.

2. Based on shared resources and anticipated services provided by Dom1, Dom1 security administrators create a set of roles R_{vo1} and assigns these roles to g_{vo1}. That is, $(g_{vo1}, r) \in GA$ only if $r \in R_{vo1}$.

3. At the same time, permission-role assignment relations are defined by Dom1 security administrators, i.e., by specifying the permissions of each role to access shared resource. Only permissions that are needed for VO1 are defined and assigned to necessary roles.

4. Dom1 registers the role names to VO1 authorization server. Typically, metadata of the roles are included for the registration, such as permission abstractions and constraints. Permission abstraction is useful to provide necessary information about the permissions of a role without revealing details, e.g., for privacy purpose of Dom1. Constraints can specify further fine-grained access control policies, such as separation of duty, prerequisite roles, and cardinality. For example, a policy can specify that a particular role only can be assigned to a limited number of users within a VO.

5. Based on VO-level security requirements, VO1 authorization server assigns roles to users. This group-level user-role assignment can be performed before or upon access requests. For example, when joins VO1, a user obtains a set of credentials which specifies its assigned roles, or its roles are determined when it generate access request to Dom1 through VO1. Role credentials can be pulled by Dom1 or pushed by user or VO1, which is determined by underlying architecture and security requirements [19]. Each VO has policies to determine how to assign roles to users, e.g., by evaluating user attributes, which are out of the scope of this paper.

6. Once roles are assigned to users, the permissions of a user can be determined by Dom1 when access requests are generated from the user to shared resources.

On a high level view, security administrators in Dom1 define resources and permissions that can be shared within VO1. Roles are defined as permission interfaces such that VO1 can leverage these information to authorize users from the community. The group-level user-role assignment is performed by VO1. Through this mechanism, the permission definitions and user-role assignments are separated such that security administrators in Dom1 can change the permissions of the roles without the involvement of VO1 and the community, according to its own policies. Also, VO1 can update user-role assignments without considering the detailed implementation of the roles and their associated permissions in Dom1.

For multiple VOs where resources are shared by Dom1, multiple groups are created by security administrators. Similarly, for multiple resource providers in VO1, each of them creates a group and determines their shared resources and

permissions to VO1 members. As these groups and permissions are managed by individual domains, they do not introduce centralized administration overhead for VO1. In general, a role is assigned with permissions from a single group such that it is provided to a single VO. In some cases, cross-group roles can be defined to enable service composition between VOs, which will be explored in our future work.

3.2 Authorization Scheme in Ad-hoc Collaborations

In ad-hoc collaborations with Grid architecture, there is no pre-established VO service such that user access control should be performed by individual domains. There are two approaches: user-role assignment by resource provider domain and by resource consumer domains. The first approach is similar to traditional centralized authorization management [2], where a user's roles are determined on the resource provider side upon access requests. The second approach can efficiently support decentralized authorization management. The following explains how our model can be applied in this approach.

1. Similar to above, upon a collaboration request, Dom1 defines resources, permissions, and roles for the collaboration between other domains. Typically, Dom1 initializes the collaboration.

2. For each participating domain (say Dom2), Dom1 creates a group g_{Dom2} and assigns roles to this group. Multiple groups can be created for different domains. Depending on security requirements, different groups can have shared roles or not.

3. Once roles are defined for g_{Dom2}, Dom1 distributes role information to Dom2. Similarly, role information can include permission abstractions and constraints that help security administrators in Dom2 to assign users. A role can be certified in different formats, such as a digitally signed credential by Dom1 or a unique token.

4. Upon receiving role credentials or tokens, security administrators in Dom2 assign users to these roles. A user-role assignment in this group level can be in another credential or token signed by Dom2. For example, Dom2 signs a role credential or a token concatenated with a local user identity together.

5. Along with an access request, a user from Dom2 sends its role credential or token to Dom1 to active a particular role. Dom1 verifies if the role is assigned by Dom1, and the credential is singed by Dom2. If both are correct, the user can obtain the permissions associated with the role, otherwise the access request is denied.

On a high level view, in ad-hoc collaboration without pre-established VO services, each domain has its own security administrators such that local users are authorized to access shared resources while the real permissions are controlled by the resource provider. Our model naturally supports this decentralized authorization management. An important feature of this scheme is that, according to global security constraints and dynamic user activities, permissions in resource provider can be dynamically updated without effect to resource consumer domains.

As aforementioned, in Grid environment, mapping user to groups (UM) is primarily managed by individual domains and organizations, according to their local policies. For example, a VO may enroll users by their memberships or skill qualifications. Our authorization scheme focuses on the separation of permission-role and user-role assignments between resource providers and other organizations. The details of UM and SUA are not covered in this paper.

4 Prototype Implementation and Evaluation

To show the feasibility and performance of our approach, we implement a prototype system by enhancing the access control module in Grid File System (GFS) [16, 25] with GB-RBAC model. This Section first gives an overview of our prototype and then presents some results of performance study.

4.1 Overview

GFS is a file sharing system using Globus Toolkit [16, 25]. In original GFS, access control is implemented by a network file system (NFS) server [5] based on the domain name of a user, where access control policies are specified by a configuration file and enforced when the file system is mounted by the user. As a complement for this coarse-grained access control, we provide a fine-grained access control which is built on GB-RBAC.

Figure 3 shows the architecture overview of our prototype based on Globus Security Infrastructure [10]. The architecture includes three main components: user platform, GFS-enabled NFS server, and authorization server. The authorization server controls user accesses in GFS. All the access control polices are stored in the server. For simplicity, in our prototype we use the same GFS server machine to run file sharing service and VO authorization service such that group-role and user-role assignment relations are both stored on the same authorization server.

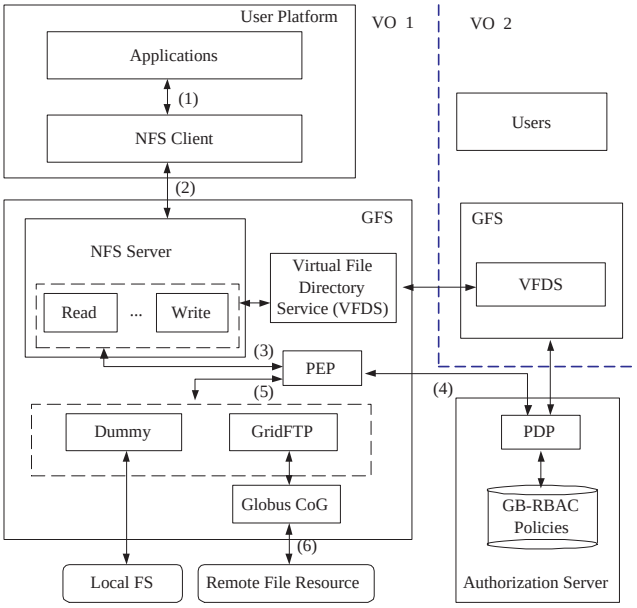

Figure 3. Prototype architecture with Grid File System (GFS)

An access session is initialized by an application on the user platform and works as follows. First the application generates an access request from the NFS client, and the request is submitted to the NFS server (step 1 and 2). A set of parameters including user role credentials, object attributes (name and path), and access action are pushed by the NFS server to the policy enforcement point (PEP) (step 3). PEP then queries the policy decision point (PDP) (step 4). After receiving the request, PDP retrieves corresponding policies from the GB-RBAC database and evaluates the access request, and returns PEP the access decision (step 4). PEP then enforces the access decision by either performing the requested action (step 5) or returns an exception. The operation may be performed on GFS's local file resource, or on remote file resource via GridFTP. The Virtual File Directory Service (VFDS) is a service that enables file sharing between VOs.

Note that although both group-role and user-role relations are stored in the same authorization server in our prototype, they are specified by different policy files so as to simulate the decentralized authorization scheme. We are going to extend our prototype with distributed policy decision points in a single VO.

The GFS server is built on a Linux-2.6.12 machine which has Centrino 1.3GHz CPU and 512MB memory, and the authorization server is built with Java 1.4.2 and works on the same machine. The user platform is built on a Fedora-2.5.9 machine which has Pentium 4 1.7 GHz CPU and 640MB memory.

4.2 Policy Specification

Our prototype uses the extensible access control markup language (XACML) [26] to specify GB-RBAC policies. Using the Sun's XACML library [1], the PDP module interprets XACML policies and makes access decisions.

As said, both permission-role and user-role policies are located in the authorization server. Figure 4 shows the skeleton of two sample policies of them, respectively.

```
<PolicySet PolicySetId="Dom1:Role:Permission">
  ...
  <Subjects>...urn:mynamespace:role:VO1_monitor...</subjects>
  <Resources> ... /log/VO1_log ... </Resources>
  <Actions> ... write ... </Actions>
  ...
</PolicySet>


<PolicySet PolicySetId="VO1:User:Role">
  ...
  <Subjects>
    <SubjectMatch>192.168.3.4</SubjectMatch>
    <SubjectMatch>urn:mynamespace:group:VO1</SubjectMatch>
  </Subjects>
  <Resources>...urn:mynamespace:role:VO1_monitor...</Resources>
  <Actions> ... membership ... </Action>
  ...
<PolicySet>
```

Figure 4. Policies for permission-role and user-role assignments in prototype

In these two policies, `VO1_monitor` is a role name in VO1. The first policy states that this role has permission to write in `/log/VO1_log` directory, which is specified by Dom1. The second policy states that the user logging from `192.168.3.3` within group VO1 is assigned with role `VO1_monitor`, which is specified by VO1. The net effect of these two policies specifies the user logging from `192.168.3.3` within VO1 has permission to write `/log/VO1_log` on Dom1.

Message integrity and authenticity between domains and organizations should be protected, e.g., with Web Services security protocols. For simplicity we do not include here.

4.3 Performance Evaluation

As a GB-RBAC decision is determined by verifying subject (requesting user) credentials, objects (resources) and actions, the performance of the system should be considered. First of all, the overhead of user authentication in our prototype is the same as that in original GFS. Therefore we only evaluate the performance when the remote application mount the GFS and typical operations enforced in the GFS with our authorization server.

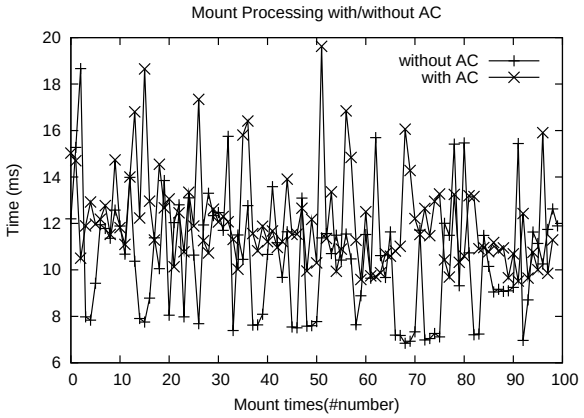

Figure 5. Mount processing with/without access control

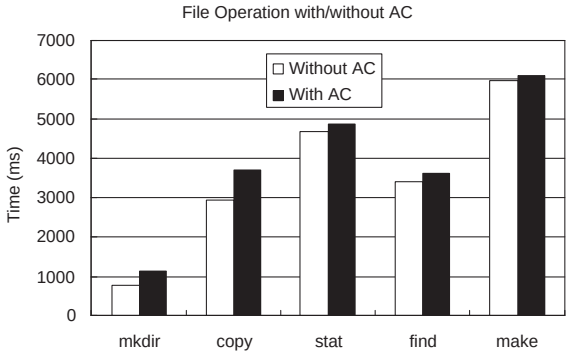

Figure 6. File operation with/without access control

According to Figure 3, access requests are generated from user platforms, and permission requests are generated in GFS server and sent to authorization server. As shown in Figure 5, the time of per *mount* event without access control (AC) ranges from 6ms to 16ms and the time of per *mount* event with AC ranges from 10ms to 19ms. The average time for *mount* event with AC is about 12.0239ms and the average time without AC is about 10.4377ms, which implies that our access control mechanism introduces about 15% extra overhead. As *mount* is a one-time operation when a remote file system is firstly used, this performance overhead is reasonable.

We further study the performance of some typical file operations after mounting. Figure 6 illustrates the results with/without AC for operations of *mkdir*, *copy*, *stat*, *find*, and *make* events, respectively. The operations are conducted as the following: *mkdir* event makes 15 new directories in GFS; *copy* event copies 70 files into the new directories; *stat* recursively states all the files in a directory of GFS which contains 140 files; *find* scans all the files in a directory and finds the specified file; *make* compiles source code files using *gcc* in a directory which contains 70 *c* files. As shown in Figure 6, the overheads introduced by authorization service in these five types of operations range from 2% to 26% more than that without AC, and only *mkdir* operation has more than 20% extra overhead. In most realworld applications, AC is only checked once during a type of continuous operations. For example, copying a batch of files to a directory only checks if the user has the permission to write the directory at the beginning. Therefore we believe that the authorization performance is acceptable for typical distribute file system accesses.

5 Related Work

Originally in many Grid systems, each resource provider uses a *grid-mapfile* to map external resource consumers to local identities and defines their permissions. As user has to be mapped to a local account, this approach is neither scalable nor flexible, especially for dynamic environment where user's permissions and activities change.

The Community Authorization Service (CAS) [20] is a centralized approach, in which a CAS server maintains the access control policies and the PDP is deployed on the CAS side. Role-based VO sub-groups approach based on CAS is proposed in [6]. Although these approaches solve the scalability problem, they lack flexibility for ad-hoc collaborations where no VO-level services are pre-established. Also, CAS lacks the flexibility to support a new resource provider which has not established service relationship with CAS, or an existing resource provider to change its policy regarding its shared resources. In our approach, on the other hand, PDP is actually separated into two levels: resource provider level and VO level, and each level manages authorizations according to local policies. Thus scalable, flexible, and autonomous authorization is achieved.

Virtual Organization Membership Service (VOMS) [3] describes an approach in which each resource provider has a set of local policies. To access shared resources, a user provides an attribute certificate issued from a VO which identifies the role, group name, and capabilities of the user. By moving the PDP from centralized server to each resource provider's local site, VOMS can solve the scalability problem with gridmap file and the flexibility of CAS, but it cannot support ad-hoc collaborations without well-established infrastructure since it still requires a (globally) centralized attribute authority. Further, since a role in VOMS defines

specific capabilities for a specific user, it cannot support decentralized authorization management, i.e., separation of user management in VOs and permission management in resource providers

PRIMA [17] is a privilege management system which supports ad-hoc collaboration and permission delegation. To submit a request to a resource provider, a user provides a set of attributes, which define the privileges of the user, such as file access permissions, user quota, and network access. The resource provider assigns permissions to the user with these attributes, according to its local policies. A shortcoming with this approach is that each resource provider takes whole responsibility of authorizations such that the access control is not efficient and scalable. Another shortcoming is that, in a dynamic collaborative environment, the privileges of a user may change according to the resource status in an resource provider, or some constraints with other concurrent jobs running in the resource provider. Therefore the pre-issued privilege attributes in PRIMA cannot support this dynamic and in-time permission assignments. However, with our approach, user are managed with roles by VO or collaborating domains and a resource provider only takes care of actual permissions, which simplifies the authorization. Further, a resource provider can change permissions associated to roles locally according to dynamic computing environment.

Akenti [27] is a distributed policy management system, where a set of stakeholders define conditions for a resource usage. An resource provider makes authorization decisions based on all these conditions in attribute certificate format. Condition certificates are pulled by PDP. The main different between this and our approach is that in our approach, each resource provider manages authorization policies on permissions to its local resources, which respects the final control of its shared resources.

PERMIS [7] is an authorization framework which uses X.509 certificate to specify authorization policies. Role is regarded as a user attribute in PERMIS, and hierarchical RBAC and delegation are supported. PERMIS focuses on the problem of general and flexible authorization architecture, where each resource provider deploys its authorization service, and policy and user certificates are either pulled by the authorization service or pushed by other authorities. However, our approach focuses on the problem of separating authority in RBAC policy definition. Therefore, policies defined with our model can be deployed and enforced with PERMIS framework.

RBAC-based approaches have been proposed for secure interoperation in multi-domain environments [13, 23], where each domain deploys RBAC policies and a set of global access control policies are composed to control shared resources accesses. A framework for secure collaboration between domains is proposed in [24], where each domain uses RBAC and policies are locally enforced by individual domains in a mediator-free manner. Our approach also leverages the management flexibility of RBAC but with an extended model enhanced with group concept. The application in Grid environment confirms that our model is efficient in authorization management.

Delegation approach has been used in ad-hoc collaborative information sharing and Grid environment [11]. In this approach, a resource owner defines rules that other parties can make authorization on behalf of the owner. Similar to our approach, policies are developed with role-based models. However, our work is based on an extended RBAC model instead of delegation rules such that decentralized authorization scheme is easier to develop and deploy with our model.

6 Conclusions and Future Work

We propose a group-based RBAC model and apply it in authorization administration of Grid environment. The main feature of our model is to leverage the control of user management by VO or collaborative domains, while preserve the final control of permissions to shared resources by resource providers. The benefits of our approach are decentralized security management and flexible and fine-grained permission control. As a result, our model can support various computing modes such as VO-managed and ad-hoc collaborations.

As mentioned in the paper, we are extending our model and authorization schema in different aspects. First of all, constraints will be included in our future work. In our model, constraints can be defined in user-role relations or group-role relations for further fine-grained access control. Secondly, we will consider cross-group roles and explore authorization management in service composition environment, which gains increasing interest since Grids have been becoming service-oriented architectures. Thirdly but not finally, context-aware constraints will be implemented such that our schema can support dynamic permissions.

References

[1] Sun's XACML implementation, http://sunxacml.sourceforge.net/.

[2] M. A. Al-Kahtani and R. Sandhu. A model for attribute-based user-role assignment. In *Proceedings of Annual Computer Security Applications Conference*, 2002.

[3] R. Alfieri, R. Cecchinib, V. Ciaschinic, L. dell'Agnellod, A. Frohnere, K. Lorenteyf,

and F. Spatarog. From gridmap-file to voms: Managing authorization in a grid environment. *Future Generation Computer Systems 21*, 2005.

[4] ANSI. American national standard for information technology- - role based access control, ANSI INCITS 359-2004, Feb. 2004.

[5] B. Callaghan, B. Pawlowski, and P. Staubach. NFS version 3 protocol specification. *RFC 1813*, 1995.

[6] S. Canon, S. Chan, D. Olson, C. Tull, and V. Welch. Using CAS to manage role-based VO sub-groups. In *Proceedings of Computing in High Energy Physics*, 2003.

[7] D.W. Chadwick and A. Otenko. The PERMIS X.509 Role Based Privilege Management Infrastructure. In *Proc. of 7th ACM Symposium On Access Control Models And Technologies*, pages 135–140, 2002.

[8] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. Richard Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3), 2001.

[9] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organization. *International Journal of Supercomputing Applications*, 15(3), 2001.

[10] Grid Security Infrastructure. http://www.globus.org/toolkit/security/.

[11] J. Jin and G-J. Ahn. Role-based access management for ad-hoc collaborative sharing. In *ACM Symposium on Access Control Models and Technologies*, pages 200–209, 2006.

[12] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. Knowl. Data Eng.*, 17(1):4–23, 2005.

[13] J. Joshi, R. Bhatti, E. Bertino, and A. Ghafoor. Access control language for multidomain environments. *IEEE Internet Computing*, pages 40–50, Novermber-December 2004.

[14] K. Keahey and V. Welch. Fine-grain authorization for resource management in the grid environment. In *Proceedings of Grid Workshop*, 2002.

[15] Q. Li, X. Zhang, S. Qing, and M. Xu. Supporting ad-hoc collaboration with group-based rbac model. In *Proceedings of the 2nd International Conference on Collaborative Computing*, Atlanta, GA, USA, 2006.

[16] Lars Lindner. Grid file system. *Master Thesis, Hasso Plattner Institute of the University of Potsdam*.

[17] M. Lorch, D. B. Adams, D. Kafura, M. S. R. Koneni, A. Rathi, and S. Shah. The prima system for privildge management, authorization and enforcement in grid environments. In *Proceedings of the 4th International Workshop on Grid Computing*, 2003.

[18] S. Oh, R. Sandhu, and X. Zhang. An effective role administration model using organization structure. *ACM Transactions on Information and System Security*, 9(2):113–137, May 2006.

[19] J. S. Park, R. Sandhu, and G. Ahn. Role-based access control on the web. *ACM Transactions on Information and Systems Security*, 4(1), 2001.

[20] L. Pearlman, V. Welch, I. Foster, and K. Kesselman. A community authorization service for group collaboration. In *Proceedings of IEEE Workshop on Policies for Distributed Systems and Networks*, 2002.

[21] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and Systems Security*, 2, 1999.

[22] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role based access control models. *IEEE Computer, 29, (2), pp.38-47, 1996*, 29(2), 1996.

[23] B. Shafiq, J. Joshi, E. Bertino, and A. Ghafoor. Secure interoperation in a multidomain environment employing RBAC poilcies. *IEEE Transactions on Knowledge and Date Engineering*, 17(11):1557–1577, Novermber 2005.

[24] M. Shehab, E. Bertino, and A. Ghafoor. Secure collaboration in mediator-free environments. In *Proceedings of the 12th ACM Conference on Computer and Communication Security*, 2005.

[25] Grid File System. http://sourceforge.net/projects/gridfs/.

[26] OASIS XACML TC. Core specification: extensible access control markup language (xacml). 2005.

[27] M. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Transactions on Information and System Security*, 6(4), 2003.