J Grid Computing (2009) 7:265–283
DOI 10.1007/s10723-009-9116-5

Role-Based Access Control in a Data Grid Using the Storage Resource Broker and Shibboleth

Vineela Muppavarapu · Soon M. Chung

Received: 29 July 2008 / Accepted: 1 March 2009 / Published online: 26 March 2009

Abstract In this paper, we propose a role-based access control (RBAC) system for data resources in the Storage Resource Broker (SRB). The SRB is a Data Grid management system, which can integrate heterogeneous data resources of virtual organizations (VOs). The SRB stores the access control information of individual users in the Metadata Catalog (MCAT) database. However, because of the specific MCAT schema structure, this information can only be used by the SRB applications. If VOs also have many non-SRB applications, each with its own storage format for user access control information, it creates a scalability problem with regard to administration. To solve this problem, we developed a RBAC system with Shibboleth, which is an attribute authorization service currently being used in many Grid environments. Thus, the administration overhead is reduced because the role privileges of individual users are now managed by Shibboleth, not by MCAT or applications. In addition, access control policies need to be specified and managed across multiple VOs. For the specification of access control policies, we used the Core and Hierarchical RBAC profile of the eXtensible Access Control Markup Language (XACML); and for distributed administration of those policies, we used the Object, Metadata and Artifacts Registry (OMAR). OMAR is based on the e-business eXtensible Markup Language (ebXML) registry specifications developed to achieve interoperable registries and repositories. Our RBAC system provides scalable and fine-grain access control and allows privacy protection. Performance analysis shows that our system adds only a small overhead to the existing security infrastructure of the SRB.

Keywords Storage Resource Broker (SRB) · Data Grid · Virtual organization (VO) · Shibboleth · Object, Metadata and Artifacts Registry (OMAR) · Role-based access control (RBAC) · eXtensible Access Control Markup Language (XACML)

V. Muppavarapu · S. M. Chung (✉)
Department of Computer Science and Engineering,
Wright State University, Dayton, OH 45435, USA
e-mail: soon.chung@wright.edu

1 Introduction

Grid has emerged recently as an integration infrastructure for the sharing and coordinated use of diverse resources in dynamic, distributed virtual organizations. A Data Grid is an architecture for the access, exchange, and sharing of data in the Grid environment. It facilitates the coordination and sharing of a large number of geographically distributed heterogeneous datasets across different domains in a controlled and secure manner

[14]. The Storage Resource Broker (SRB) [5, 37] was developed by the San Diego Supercomputer Center (SDSC) as a logical distributed file system based on client–server architecture. The SRB can be used as a Data Grid management system as it has features to support the management, collaboration, controlled sharing, publication, replication, transfer, attribute-based organization, data discovery, and preservation of distributed data.

The SRB stores and manages the access control information of individual users in the Metadata Catalog (MCAT) database. However, because of the specific MCAT schema structure, this information can only be used by the SRB applications. If an organization has many non-SRB applications, each with its own storage format for user access control information, they cause substantial administration overhead because the system administrator has to mange them independently. Thus, multiple updates may be required if the access control information of a current user changes; and if new users are allowed to access many data resources, it can create a scalability problem with regard to administration. Additionally, as the access control information of each Grid user is maintained in the MCAT database, if users join/leave the organizations frequently, updating their access control information on an individual basis would be a difficult task. Moreover, a user may not want to reveal his/her personal information; hence, privacy protection becomes an important issue. The current identity-based access control mechanisms do not protect the privacy of the users. In our research, we developed a new access control mechanism which is not only scalable but also protecting the privacy of users.

Role-based access control (RBAC) [10, 38, 39, 41] is an approach where permissions are assigned to roles and roles are assigned to users, thereby acquiring the roles' permissions. Usually roles are created and assigned to users based on their job functions, and RBAC is more scalable than identity-based authorization because authorization information is associated with roles, not with individual users. In this paper, we propose a new RBAC system for heterogeneous data resources in the SRB by using Shibboleth [8] and the Object, Metadata and Artifacts Registry (OMAR) [31] as the main components.

Shibboleth is an attribute authorization service and is designed to provide the user attributes to the resources for access control, and it mainly targets the internet-based resources. Shibboleth supports the authentication of user at his/her home institution. By using Shibboleth, a user can either be authenticated at his/her home institution first and then present his/her authentication information and attributes to a Data Grid service, or contact a Data Grid service which is then redirected to the user's home institution to obtain the authentication information and attributes of the user. In either case, the user does not need to be authenticated at each individual site, and the individual sites do not need to maintain all the user information, as the authentication is handled by the user's home institution.

In Data Grids, users belonging to different organizations try to use the data resources and applications. If the SRB server can recognize the credentials issued by the user's home institution, then the SRB server does not need to individually manage the authentication information and attributes of each user. By using Shibboleth, the SRB server just needs to negotiate with the user's home institution regarding the security policies and then trust the authority that issues the credentials of the user. The authentication would then be handled by the user's home institution, and the required user information would be transferred to the SRB server. This approach would improve the interoperability among organizations, facilitate their collaboration, and relieve the SRB administrator from the burden of individually maintaining the authentication information of the users. Also, by using Shibboleth, access control policies can be specified based on the specific attributes of the user, such as role name and affiliation, instead of the identity of the user.

Shibboleth issues user attributes in the form of Security Assertion Markup Language (SAML) [26] assertion. In our access control system, this SAML assertion is embedded into the user's proxy credential. The SRB server can then obtain the SAML assertion containing the role names of the user in addition to his/her identity or distinguished name (DN) from the proxy credential. If a system wishes to perform authentication using the individual identity or DN of the user, it can

do so as the proxy credential contains the DN of the user. In our system, Shibboleth is also used as a Role Enablement Authority (REA), which is responsible for assigning roles to users and for activating roles within the user's session [27].

We used the Core and Hierarchical RBAC profile of the eXtensible Access Control Markup Language (XACML) [27] to specify the access control policies. XACML is a standard of the Organization for the Advancement of Structured Information Standards (OASIS) for describing the access control policies uniformly across different security domains [28]. Our system is based on the Core and Hierarchical components of the ANSI-RBAC [43], and the Core and Hierarchical RBAC profile of XACML defines how they can be specified in XACML. For the distributed storage and administration of XACML policies and the user-role assignments, we used the OMAR. OMAR is an open-source implementation from the freebXML.org, and it provides an implementation of the OASIS e-business eXtensible Markup Language (ebXML) registry specifications [25]. A registry is an information system that securely manages any content type and the standardized metadata that describes the content. The ebXML registry specifications are developed to achieve interoperable registries and repositories with an interface that enables submission, query and retrieval [25].

Our system is scalable in terms of the number of access requests as well as the number of users; and it is robust as there is no single point of failure. It supports the management of privileges and fine-grain attribute release policy; and it provides privacy protection for users. It can support a wide range of security policies using role-privileges, role hierarchies, and timing constraints. The MCAT database needs to maintain only the mapping information from Grid user roles to local roles and local policies, thus their administration overhead is reduced. When users join/leave, the MCAT administrator does not have to bother about individually adding/removing their information in the MCAT database, because OMAR can be used to directly grant/revoke the users' memberships on the roles.

We have implemented our proposed RBAC system and analyzed its performance. A Shibboleth interface program (SIP) has been designed to obtain the user's attributes from the Shibboleth service and to create the user's Shibboleth proxy credential. The SRB client has been modified to send the user's Shibboleth proxy credential to the SRB server, instead of the proxy credential. The SRB server has been modified to obtain the user's attributes from the user's Shibboleth proxy credential and then to map the user's VO role to a local role by using the corresponding information in the MCAT database. Our performance analysis shows that the proposed system incurs a small overhead in setting up the security between the client and server. This overhead is quite acceptable when we consider the benefits of our system, such as scalability in managing access control policies and reduced administration overhead for the resource providers.

This paper is organized as follows: Section 2 contains background information. In Section 3, we propose a RBAC system using Shibboleth in the SRB, and also show how to manage access control policies using XACML and OMAR. Section 4 describes the implementation details, and Section 5 describes the results of performance analysis. Section 6 contains some conclusions.

2 Background

2.1 Need for Role-Based Access Control (RBAC) in Grids

In Grids, both users and resources are dynamic. Furthermore, those users and resources may belong to multiple organizations with their own diverse security policies and mechanisms. Participating organizations may have different security models. It is important for these models to interoperate based on different levels of trust and contexts. Trust should be established not only among users and resources, but also among the resources themselves, so that they can be coordinated. These trust domains can span across multiple organizations and must adapt dynamically as participants join or leave and as resources are accessed or released [45]. Furthermore, to provide fine-grain security services, the current contexts of

the users and resources should be considered. The resource providers must understand and support the mechanisms and policies that are not strictly under their control, in order to grant or deny access to their resources.

It is desirable to group users and resources that need to be coordinated for a common goal into virtual organizations (VOs). The key requirement is to design access control mechanisms for these VOs, which can interoperate with existing local security infrastructures and allow the resource providers to have the ultimate control over their resources. In addition to accommodating multiple security mechanisms of different sites, there are a number of requirements to be met for Grid security, such as scalability, fine-granularity, delegation of rights, identity federation for users and resources, and privilege management.

Supporting role-based access control (RBAC) is desirable in Grids. RBAC shows clear advantages over traditional discretionary and mandatory access control models in such environments, because it allows a uniform representation of diverse security policies [18]. In RBAC, permissions are associated with roles, and users are assigned appropriate roles, thereby acquiring the roles' permissions [38]. In a VO with a large number of users, we could think of several groups of users, each with different level of access privileges via VO roles. When a VO role is mapped to a local role by a resource provider, it will specify the privileges a user can have; for example, access to a specific table of a database.

In VOs, users may be assigned specific tasks, and there may be certain constraints related to the execution of those tasks. For example, a user may have access to a database only during certain days of the week, or cannot execute two or more tasks at the same time. RBAC can support a wide range of security policies using role privileges, role hierarchies, and constraints.

The typical identity-based authorization used today is not scalable because authorization information should be maintained for each user. In RBAC, authorization information is associated with roles, not with individual users. It has been shown that the cost of administering RBAC is proportional to $U + P$ per role, where U is the number of individuals in a role and P is the number of permissions required by the role, while the cost of associating users directly with permissions is proportional to $U \times P$ [11].

RBAC is distinguished by its inherent support for the Principle of Least Privilege [22]. The Principle of Least Privilege requires that a user be given no more privileges than necessary to perform a job [10]. It can be easily enforced by first identifying the roles in an organization correctly and then assigning only those privileges to each role that allow the role members to perform their tasks.

Several extensions of the RBAC model have been proposed for dynamic coalition environments [9, 15, 20]. However, most of these models come with their own specifications and implementation frameworks and are not integrated with any of the existing authorization services available for Grids. Our proposed RBAC system uses the XACML standard to specify the access control policies, and is integrated with the widely used Shibboleth authorization service.

2.2 The Storage Resource Broker (SRB)

The Storage Resource Broker (SRB) is developed as a client–server middleware that provides uniform interface to access heterogeneous distributed storage resources including file systems, database systems and archival storage systems [5]. The SRB presents the user with a single file hierarchy for the data distributed across multiple storage systems. The SRB provides the following: (1) a logical namespace to describe storage systems, digital file objects and collections; (2) specific features for digital libraries, persistent archive systems and collection management systems; (3) capabilities for storing replicas of data, authenticating users, controlling access to documents and collections, and auditing accesses; (4) user-defined metadata management at the collection level and object level, and search capabilities based on the metadata [36].

The SRB, in conjunction with the Metadata Catalog (MCAT) database, supports location transparency by providing access to data resources based on their attributes rather than their names or physical locations. The MCAT database

contains file attributes, metadata, user information, security and access control information, and also maintains audit trail on data and collections of data. The architecture, entities and the attributes of MCAT database are described in [23].

2.2.1 The SRB Architecture and its Current Access Control System

The SRB system is comprised of three major components: MCAT database, SRB servers and SRB clients; and its architecture is shown in Fig. 1.

Usually a SRB system, called a SRB zone, consists of a single MCAT and one or more SRB servers and clients. When data collections and collaborations grow large with many users spread geographically, multiple SRB zones, each with its own MCAT, can form a federated SRB system for additional performance and reliability. The SRB server linked to the MCAT database is referred to as MCAT-enabled SRB server (MES server), and other SRB servers are referred to as non-MCAT-enabled SRB servers (non-MES servers).

The SRB supports two types of authentication: the ENCRYPT1, which is a password-based authentication system, and the Grid Security Infrastructure (GSI). The GSI uses public key infrastructure (PKI) mechanisms which operate on the credentials issued to each Grid user and service. The credentials, formed by an X.509 certificate, and the associated public/private keys are issued by a certificate authority (CA) trusted by all entities in the Grid. The user passwords for ENCRYPT1 and the distinguished names (DNs) for GSI are stored in the MCAT database.

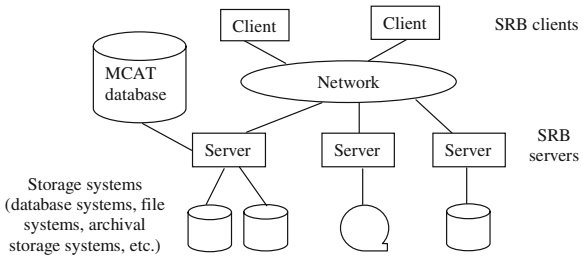

Fig. 1 SRB system architecture

The current access control system of the SRB uses Unix-like permissions and a ticket mechanism [5]. Owners of the data objects can specify which users and user groups can have what permissions (read, write, etc.) on their data objects. For all the data objects exposed by an SRB server, permissions assigned to an individual user/user group should be stored in the MCAT database. If the privileges of a user change in an organization, the corresponding information should be updated in the MCAT database.

The ticket mechanism of SRB is a restricted form of read privilege [6]. The user with the privilege of granting a ticket on a data object can create tickets and issue them to other users. This ticket could be valid for a specific period of time or for a certain number of use, thus the read access to a data object can be controlled. This ticket mechanism can only control read accesses, thus it limits the delegation of access privileges. Furthermore, the current access control system of the SRB is not scalable in a Grid environment, especially if the resource providers and users belong to multiple organizations, because it would be very difficult to manage the access control information for individual users as well as their tickets. Most of all, the access control information stored in the MCAT database cannot be used by non-SRB applications.

2.3 Shibboleth

Shibboleth [8] is an attribute authorization service developed by the Internet2 community for cross-organization identity federation [46]. Shibboleth provides the user attributes to the requested resources by using the Security Assertion Markup Language (SAML) [26] attribute statements. Shibboleth creates a distributed authorization infrastructure for Web resources, and simplifies the access control policies and makes them more scalable [4].

Privacy protection and cross-domain auditing are significant challenges in Grids [17]. An audit mechanism is responsible for producing records which track security related events [24]. For this purpose, it is essential to keep a log of the access requests and the enforced security policies. The identity of the user, such as user's DN, can be

used for identifying the user who initiated the access request. The access request can be logged at the resource together with the mapping information and the subsequent actions performed. This information can be used to find patterns that fit the profile of a system intrusion or the activities that do not fit the profiles of legitimate users [12]. However, this information could affect the privacy of users. For example, by examining the information logged at various sites for the users belonging to a particular research group, it is possible to infer their data access patterns and thus obtain information about their work.

Shibboleth allows pseudonymous interaction between users, thus protecting individual privacy while still providing basic security for the resource providers [46]. The Shibboleth service maps a user name and attributes onto a unique identifying handle. To protect the user's privacy, the service can restrict the information about the holder of the handle, depending on who is asking for the information. For example, it does not release the user's name except to those requestors who have been authorized [17]. A user could be issued a set of pseudonym identities [46], and he/she could be authenticated at different sites with different identities. The information that binds the set of pseudonym identities to the handle should be maintained securely and can be used when security violations occur.

The Shibboleth Identity Provider (IdP), which is responsible for providing the authentication information and attributes of the user, is entirely written in Java and runs in a servlet container. So, any Web service or Web application running on a Web server can contact the Shibboleth system on behalf of the client (or user).

2.4 Related Work

Shibboleth has certain advantages, such as privacy protection, over other authorization services for Grids, such as the Virtual Organization Management Service (VOMS) [1], Akenti [44], Community Authorization Service (CAS) [33], and PERMIS [32].

VOMS supports user-role assignments and provides attribute certificates, containing user's VO roles and group memberships, to the users. The attribute certificates do not provide rights directly, and they need to be interpreted by the resources. Akenti does not assign roles or group memberships to the users. Instead, it assigns the permissions to individual users directly, which is not scalable [1]. Furthermore, Akenti does not provide support for dynamic delegation [44]. Delegation is a key issue in a VO, wherein a set of rights can be delegated to a program for it to act on behalf of a user. A program should also be able to delegate some of its rights to other programs [13]. CAS issues authorization assertions containing the list of actions the user is allowed to perform on a resource, and the resource providers are allowed to delegate some of the authority for maintaining fine-grain access control policies to communities. However, CAS is a centralized authorization system, so it could be a bottleneck in terms of performance.

PERMIS [32] is an attribute-based authorization service, like VOMS. They use assertions that bind the attributes to users for authorization, as opposed to the typical identity-based authorization [16]. However, currently they do not support any standard for transferring attributes from the attribute authority to the Grid services and for expressing the policy regarding those attributes [16].

2.5 Our Previous Work

In [34], we described how the Community Authorization Service (CAS) can be used to enhance the security mechanism in Open Grid Services Architecture – Data Access and Integration (OGSA-DAI) [2, 3]. In OGSA-DAI, access control causes substantial administration overhead for the resource providers in VOs, because each of them has to manage a role-map file containing the authorization information of individual Grid users. To solve this problem, we used the CAS provided by the Globus Toolkit to support the RBAC within OGSA-DAI framework. CAS grants the membership on VO roles to users. The resource providers then need to maintain only the mapping information from VO roles to local database roles in the role-map files, so that the number of entries in the role-map file is reduced dramatically. In [35], we described how RBAC

policies such as role privileges, constraints and hierarchies can be specified and managed for a community using CAS.

Our access control methods proposed in [34, 35] provide increased manageability for a large number of users and reduce day-to-day administration tasks of the resource providers, while they maintain the ultimate authority over their resources. However, a single CAS server can be a bottleneck if a large number of users attempt to access it at the same time, and it can be a single point of failure. Also, those access control methods do not provide privacy protection for the users because every CAS credential contains information that identifies the user. They do not support the diverse policies across domains, either.

3 A RBAC System with Shibboleth for the SRB

In this section, we propose a new RBAC system using Shibboleth for the SRB system. The Core and Hierarchical RBAC profile of the eXtensible Access Control Markup Language (XACML) [27] is used to specify the access control policies. For the distributed storage and administration of XACML policies, including user-role assignments, we used the Object, Metadata and Artifacts Registry (OMAR).

3.1 Drawbacks of the Existing Access Control Method in the SRB

The SRB server authenticates the user based on his/her individual identity. Upon successful authentication, the SRB server checks whether that identity is authorized to perform the requested action, such as accessing a file, creating a new file/directory, etc. However, this method is not suitable when the number of users is large in a SRB system, because the MCAT database server could become a bottleneck. Moreover, when a local security policy is changed, it may require the MCAT database to be updated.

To illustrate this access control problem in a SRB system, let us consider the following example: Suppose that a university created a Data Grid by installing an MCAT database and an SRB server at the university level. For simplicity, let us assume that certain data resource in this SRB system can be accessed only by the students of the Computer Science department.

With the current implementation, the MCAT database administrator has to manage all the identities of the students to allow the access to the data resources. If any student joins/leaves/changes the department, his/her identity has to be added/deleted/updated in the MCAT database. Considering the frequency of those personnel changes, the MCAT database has to be updated quite often, and it increases the burden on the administrator. If the university decides to form a federated SRB system with other universities, the administration overhead would increase multifold.

3.2 Our System Architecture with Shibboleth

Our idea is to use the VO roles of the users instead of their identities for access control, so that the MCAT database needs to maintain only a few roles, instead of all the user identities. This reduces the administration overhead of the MCAT administrator as he/she does not have to worry about individually adding/removing the user identities in the MCAT database.

In our system, Shibboleth maintains the security policies of the VOs, grants the users' memberships on VO roles, and then authorizes them to use the privileges of those roles. The data resource providers only need to maintain the mapping information from the VO roles to the local roles and local policies, thereby reducing the number of entries to be maintained in the MCAT database dramatically. Our system also allows the specification of policies at the VO level, thus if the users do not possess the required privileges, their access can be denied at the VO level. This eliminates unnecessary authentication, mapping and connection overheads on the resource providers. When users join/leave the VO, Shibboleth can just grant/revoke their memberships on the VO roles.

Our system architecture is shown in Fig. 2. Shibboleth releases the attribute values related to the user. The SRB server obtains the user attributes released by Shibboleth, and then makes

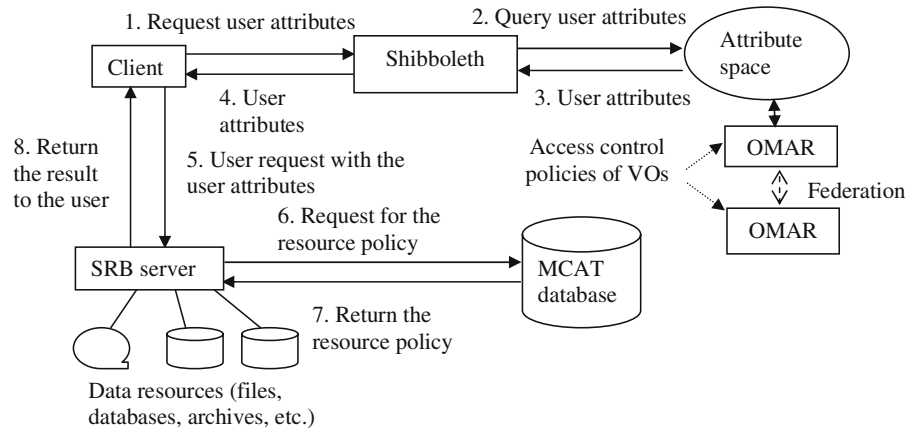

Fig. 2 Proposed architecture

an authorization decision based on the user attributes and the corresponding resource policy information. Based on its decision, the SRB server either performs or denies the client's request.

The data flow in our system is described in detail as follows:

1. The client sends an attribute query (as a SAML request) to Shibboleth.
2. To obtain the user information, Shibboleth queries the OMAR registry, which stores the user-role assignments and the access control policies in XACML.
3. The user attributes are returned from the OMAR registry.
4. Shibboleth sends the SAML response containing the user attributes to the client.
5. The client makes an access request to the SRB server along with the user attributes.
6. The SRB server obtains the user attributes from the client, and then queries the MCAT database to obtain the corresponding resource policy information, including the mapping of the VO role to local roles and their access permissions.
7. The MCAT database returns the resource policy information to the SRB server.
8. If the resource policy allows the access, then the client request is performed by the SRB server, otherwise it is denied.

Our system works in the push model, as the user can directly obtain the permissions from the authorization service and can pass them to the SRB server at the time of making a request. The SRB server then verifies the authenticity of the user and authorizes the user based on the permissions obtained, provided the authority that issued them is trustworthy. An advantage of the push model is that it allows the user to explicitly select a role from the set of roles he/she is entitled to. Additionally, the push model is inherently scalable because the SRB server does not need to contact the authorization service to obtain the user attributes for each access request. In case that the user and the authorization service belong to the same organization and are protected by a firewall, the push model should be deployed because the SRB server may not be able to contact the authorization service directly.

Shibboleth also supports the pull model, where the user directly contacts the SRB server first. The SRB server then contacts Shibboleth to obtain the user's permissions. An advantage of the pull model is that it can be deployed easily because users do not need to interact with the authorization service [46]. However, considering the advantages of the push model, our system is currently deployed in the push model alone.

In traditional systems, an organization's entire security policy is managed by a single central authority; however, in a VO, its security policy is distributed across its member organizations. Thus, it may be necessary to manage the user attributes across different organizations and to aggregate them for an authorization decision. Shibboleth allows the VO to manage different subsets of its attribute space at different member organizations,

as each member organization operates a Shibboleth service. In addition, the local access control policies of each organization can be managed by OMAR, which also allows the federation of the local access control policies of different organizations. Thus, distributed administration of the access control policies of a VO is feasible, while maintaining the local autonomy of each member organization. This also ensures the privacy of users, as the individual user information is solely administered at the user's home organization.

3.3 eXtensible Access Control Markup Language (XACML)

eXtensible Access Control Markup Language (XACML) is an OASIS standard for the representation of access control policies uniformly across diverse security domains in a flexible, extensible XML format [28]. XACML policy language defines the following main components to represent the policies:

(1) A <PolicySet> contains a set of access control policies or other policy sets.
(2) A <Policy> represents an access control policy described through a set of rules.
(3) A <Rule> represents an access rule or permission.

An XACML <PolicySet>, <Policy> or <Rule> may contain a <Target> element. A <Target> element specifies a set of subjects, resources, actions and environments to which the <PolicySet>, <Policy> or <Rule> is applied [28].

The Core and Hierarchical RBAC profile of XACML specification defines how ANSI core and hierarchical RBAC standard [43] can be specified in XACML. The Core and Hierarchical profile further defines the following components:

(1) Permission <PolicySet> (PPS) contains <Policy> elements and <Rules> associated with a given role. A PPS may also contain references to other PPSs associated with other roles that are junior to the given role, thereby allowing the role to inherit all the permissions associated with its junior roles. The <Policy> elements and <Rules> of the PPS describe the resources and the permissions on the resources along with any conditions on those permissions.
(2) Role <PolicySet> (RPS) associates a role with the corresponding PPS. Each RPS can only reference a single PPS.
(3) Role Assignment <Policy> (or <PolicySet>) defines which roles can be enabled or assigned to which subjects.

3.4 Specification of RBAC Policies Using XACML

The specification of RBAC policies for the VO roles can be done by the VO administrator using the XACML policy language. A single policy can include any number of decentralized rules, each managed by different organizations [21]. In multi-domain environments such as VOs, it is necessary to distinguish the user attributes in a VO from the attributes of other VOs. For example, the employee role in the Accounting subgroup of the VO Alpha may have different permissions from the employee role in the Accounting subgroup of another VO Beta. Hence, when making an authorization decision, it is important to know not only the role name employee but also the VO name, either Alpha or Beta.

In order to manage and identify the attributes from different domains, Shibboleth uses scoped attributes, defined in SAML, which can include the domain name. A scoped attribute is a combination of a value and its scope. Scope identifies the domains and sub-domains in which the values are defined. An example scoped attribute is "faculty@abcuniv.edu", which identifies the value "faculty" in the scope "abcuniv.edu". However, the XACML profile does not support these scoped attribute values for subjects such as roles. Mapping SAML to XACML allows the systems using XACML to store SAML attributes [30].

We can show how the scoped attribute values can be specified in the RBAC profile of XACML to represent the role names specific to VOs and VO subgroups. For example, the employee role in the Accounting subgroup of Alpha can be represented in XACML as shown in Fig. 3. The VO and subgroup names, "Alpha. Accounting", represent the scope of the employee

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicySetId="RPS:employee:role" PolicyCombiningAlgId="&policy-combine;permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;anyURI-equal">
          <AttributeValue DataType="&xml;anyURI" Scope="Alpha.Accounting">&roles;employee</AttributeValue>
          <SubjectAttributeDesignator AttributeId="&role;" DataType="&xml;anyURI"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Fig. 3 RPS of employee role

role. The RPS of the employee role, shown in Fig. 3, references the PPS of the employee role via <PolicySetIdReference>. The PPS of the employee role is shown in Fig. 4. The RPS of the manager role is not shown here, but is similar to the RPS of employee, except that the role name is

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicySetId="PPS:employee:role"
PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Policy PolicyId="Permissions:specifically:for:the:employee:role" RuleCombiningAlgId="&rule-combine;permit-overrides">
  <Rule RuleId="Permission:to:read:data:from:DataSet1" Effect="Permit">
   <Target>
    <Resources>
     <Resource>
      <ResourceMatch MatchId="&function;string-equal">
       <AttributeValue DataType="&xml;string">DataSet1</AttributeValue>
       <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
      </ResourceMatch>
     </Resource>
    </Resources>
   <Actions>
    <Action>
     <ActionMatch MatchId="&function;string-equal">
      <AttributeValue DataType="&xml;string">read</AttributeValue>
      <ActionAttributeDesignator AttributeId="&action;action-id" DataType="&xml;string"/>
     </ActionMatch>
    </Action>
   </Actions>
  </Target>
  <Condition>
   <Apply FunctionId="&function;and">
    <Apply FunctionId="&function;time-greater-than-or-equal">
     <Apply FunctionId="&function;time-one-and-only">
      <EnvironmentAttributeDesignator AttributeId="&environment;current-time" DataType="&xml;time"/>
     </Apply>
     <AttributeValue DataType="&xml;time">9h</AttributeValue>
    </Apply>
    <Apply FunctionId="&function;time-less-than-or-equal">
     <Apply FunctionId="&function;time-one-and-only">
      <EnvironmentAttributeDesignator AttributeId="&environment;current-time" DataType="&xml;time"/>
     </Apply>
     <AttributeValue DataType="&xml;time">17h</AttributeValue>
    </Apply>
   </Apply>
  </Condition>
 </Rule>
</Policy>
</PolicySet>
```

Fig. 4 PPS of employee role

manager and the <PolicySetIdReference> references PPS:manager:role shown in Fig. 5.

A policy could contain the individual permissions and timing constraint information of a role, indicating that the user in that role can access a resource during a specific time period. In [19], the authors introduced a specification of the periodicity and duration constraints in RBAC. Our system can use this specification, and the timing constraints can be represented in the XACML policy by using the Environment attributes. For example, Fig. 4 contains the periodicity of the time duration between 9:00 AM and 5:00 PM each day. The PPS of the employee role grants the permission to execute the read operation (specified within <Action>) on the resource 'DataSet1' (specified within <Resource>) only from 9:00 AM to 5:00 PM (specified within <Condition>). The PPS of the manager role, shown in Fig. 5, grants the permission to execute the write operation on 'DataSet1'. It references the PPS of the employee role via <PolicySetIdReference>, thereby inherits all the permissions of the employee role.

3.5 Object, Metadata and Artifacts Registry (OMAR)

For the storage and management of the XACML policies and the user-role assignments, we used the Object, Metadata and Artifacts Registry (OMAR) [31] which provides an implementation of the OASIS ebXML registry specifications. The ebXML specifications are developed to achieve interoperable registries and repositories, with an interface that enables submission, query and retrieval of the registry and repository contents [25]. An ebXML registry is an information system that securely manages any content type and the standardized metadata that describes the content. It also provides a set of services for sharing of its content and metadata between organizations in a federated environment [29].

OMAR stores data in a repository and stores the associated metadata as registry objects [29]. The relationship between registry objects is represented by an association object. OMAR allows many-to-many associations between registry

Fig. 5 PPS of manager role

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="PPS:manager:role"
PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Policy PolicyId="Permissions:specifically:for:the:manager:role"
RuleCombiningAlgId="&rule-combine;permit-overrides">
  <Rule RuleId="Permission:to:write:data:to:DataSet1" Effect="Permit">
     <Target>
      <Resources>
       <Resource>
        <ResourceMatch MatchId="&function;string-equal">
         <AttributeValue DataType="&xml;string">DataSet1</AttributeValue>
         <ResourceAttributeDesignator AttributeId="&resource;resource-id"
DataType="&xml;string"/>
        </ResourceMatch>
       </Resource>
     </Resources>
    <Actions>
       <Action>
            <ActionMatch MatchId="&function;string-equal">
                    <AttributeValue
            DataType="&xml;string">write</AttributeValue>
                    <ActionAttributeDesignator AttributeId="&action;action-id"
            DataType="&xml;string"/>
            </ActionMatch>
      </Action>
     <Actions>
    </Target>
   </Rule>
  </Policy>
  <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

objects. OMAR uses an object, called slot, to add attributes dynamically to registry objects.

3.6 Administration of the RBAC Policies in XACML by Using OMAR

The XACML role policy information of individual users can be managed by the OMAR. OMAR stores the XACML policies in their entirety as repository items in a relational database and classifies them as either XACML Policy object or XACML PolicySet object. However, as policies are stored in their entirety, any updates in policies become difficult especially for VOs whose policies are complex and tend to change dynamically. To solve this problem, we propose to split each policy into components and store them as different objects, as shown in Fig. 6.

OMAR supports the federation of registries by defining a federation object and by creating an association between the registry objects and the federation object. Federation allows multiple registries to link together seamlessly and appear as a

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicySetId="PPS:employee:role"
PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Policy PolicyId="Permissions:specifically:for:the:employee:role" RuleCombiningAlgId="&rule-combine;permit-overrides">
  <Rule RuleId="Permission:to:perform:read:on:DataSet1" Effect="Permit">
    <Target>
      <Resources>….DataSet1… </Resources>
      <Actions>…read… </Actions>
    </Target>
  </Rule>
 </Policy>
</PolicySet>
```

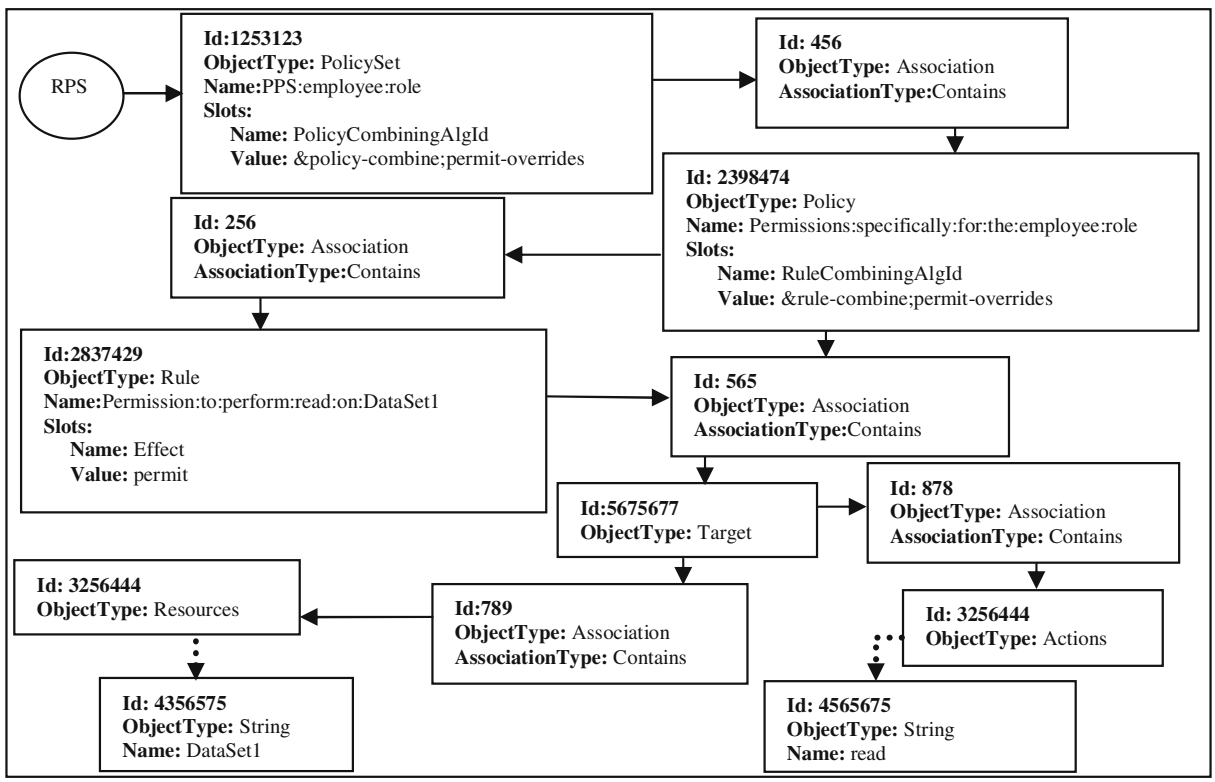

Fig. 6 A part of employee PPS and corresponding storage in OMAR

single logical registry, while retaining local autonomy and security. Thus, XACML-based policies can be stored and managed across multiple sites in the Grid, while each site maintains its own registry and Shibboleth service. Hence, there is less chance to have a bottleneck, and there is no single point of failure in the system as attribute queries are distributed among the Shibboleth services.

One of the key features of RBAC is the ability to manage itself through administrative roles and permissions [40]. Users in administrative roles can create roles and role hierarchies; make user-role and permission-role assignments; and specify constraints. Furthermore, they can grant administration privileges to other users. This administration feature of RBAC is not directly addressed by the RBAC profile of XACML, but can be easily realized through OMAR.

A user with OMAR registry administration privileges can create policy objects and determine how other registry users can access them through the Access Control Policy (ACP) file. In particular, that user can create a registry role (by creating policy objects) and assign the privilege of creating/accessing certain registry objects to that role. A registry role is an OMAR component similar to a role related to a database. The user who creates a registry role can also grant memberships on that registry role to other registry users. For example, a VO administrator having OMAR registry administration privileges can create a new registry role and assign (to that role) only the privilege of creating users in VO employee role. The VO administrator can then assign a VO manager to the new registry role (after creating a registry account for the VO manager). Then, the VO manager can create new users in VO employee role by himself/herself.

4 Implementation Details

Our RBAC system with Shibboleth supports the push model, where the user directly obtains his/her authorization information, in terms of VO roles, from Shibboleth and passes them to the SRB server at the time of making a request. We have designed a new program, called *Shibboleth interface program* (SIP), which can be invoked by the user to obtain the authorization information from Shibboleth. This program then creates a Shibboleth proxy credential by including the authorization information into the user proxy credential. The user proxy credential is formed by an X.509 certificate and the associated public/private keys. The X.509 certificate is issued by a certificate authority (CA) trusted by all entities in the Grid [7]. As the user proxy credential is valid for a limited period of time, 12 hours by default, so is the Shibboleth proxy credential. The SRB client delegates the Shibboleth proxy credential to the SRB server to request an access to a data resource. The SRB server-side has been extended to parse the user attributes in the new Shibboleth proxy credential to obtain the VO role of the user.

4.1 Client-Side Implementation

The SIP utilizes the Java GridShib SAML tools [42], capable of handling SAML attribute requests and responses. As shown in Fig. 7, the SIP invokes the SAML attribute query client and passes the user proxy credential to it. The attribute query client then contacts Shibboleth at the designated URL. Shibboleth retrieves the user's attributes from OMAR and returns them in the form of a SAML assertion to the SIP. The SIP also

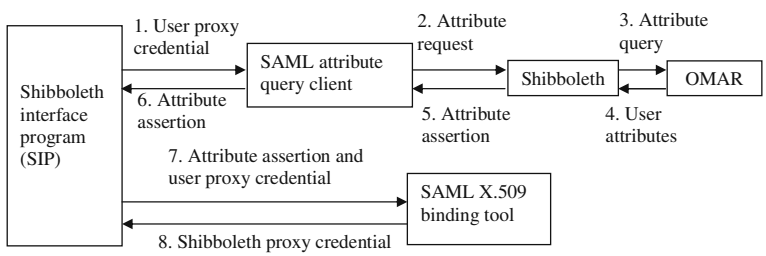

Fig. 7 Data flow involving the Shibboleth interface program (SIP)

utilizes the SAML X.509 binding tool to bind the SAML assertion as a non-critical extension to the user proxy credential. The new credential is called Shibboleth proxy credential.

Figure 8 shows an example Shibboleth proxy credential which contains a SAML assertion issued by Shibboleth. The assertion, italicized in Fig. 8, is valid for a limited period of time specified by the *Conditions* statement in the assertion. The Shibboleth proxy credential is stored as a default temporary file, which can be used by any Grid applications, including the SRB client.

4.2 Server-Side Implementation

The SRB server has been modified to obtain the VO role of the user from the Shibboleth proxy credential. The SRB server parses the SAML assertions to obtain the user role based on the algorithm shown in Fig. 9.

The SRB server tries to retrieve the user's certificates from the credential presented. If a certificate exists, then the server tries to obtain the non-critical extension containing the SAML assertions from it. The Object Identifier (OID) of that

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 867284 (0xd3bd4)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: O=Grid, OU=GlobusTest, OU=simpleCA-xxxx.cs.wright.edu, OU=cs.wright.edu, CN=uuuu, CN=613767748
    Validity
      Not Before: Apr 30 00:16:16 2008 GMT
      Not After : Apr 30 12:21:16 2008 GMT
    Subject: O=Grid, OU=GlobusTest, OU=simpleCA-xxxx.cs.wright.edu, OU=cs.wright.edu, CN=uuuu, CN=613767748, CN=67935642
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (512 bit)
        Modulus (512 bit):
          00:8d:cd:e0:1f:b9:57:dc:a9:3a:78:0a:4d:03:64:
          2c:94:42:c6:ca:2f:33:b1:71:9f:49:1c:29:2d:73:
          dd:8f:1e:10:5d:c0:ef:26:dc:ce:fc:18:9c:f9:60:
          e4:11:5a:38:7d:76:63:ec:34:50:b0:e9:38:af:5f:
          b0:21:a4:2c:a7
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      1.3.6.1.4.1.3536.1.222: critical
        0.0
..+.....
        1.3.6.1.4.1.3536.1.1.1.10:
          <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" AssertionID="_1063ef80756c76c747609152bbb4b567" IssueInstant="2008-04-
30T00:21:16.283Z" Issuer="https://idp.example.org/shibboleth" MajorVersion="1" MinorVersion="1"><Conditions NotBefore="2008-04-
30T00:21:16.283Z" NotOnOrAfter="2008-04-30T00:51:16.283Z">
<AudienceRestrictionCondition><Audience>https://globus.org/gridshib</Audience><Audience>urn:mace:gridshib:metadata:sp</Audience></
AudienceRestrictionCondition></Conditions><AttributeStatement><Subject><NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName" NameQualifier="https://idp.example.org/shibboleth">CN=uuuu,OU=cs.wright.edu,OU=simpleCA-
xxxx.cs.wright.edu,OU=GlobusTest,O=Grid</NameIdentifier></Subject>
<Attribute AttributeName="urn:mace:dir:attribute-def:role" AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">
<AttributeValue>manager</AttributeValue></Attribute>
<Attribute AttributeName="RolePermissions" AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">
<AttributeValue>Permission:1:Resource:DataSet1</AttributeValue><AttributeValue>Permission:1:Action:read</AttributeValue>
<AttributeValue>Permission:2:Resource:DataSet2</AttributeValue><AttributeValue>Permission:2:Resource:DataSet3</AttributeValue>
<AttributeValue>Permission:2:Action:read</AttributeValue>><AttributeValue>Permission:2:Action:write</AttributeValue></Attribute>
</AttributeStatement></Assertion>
  Signature Algorithm: md5WithRSAEncryption
    4f:82:83:29:38:ef:ef:bf:60:49:94:70:36:67:3d:e7:4b:5f:
    a1:46:b5:3b:e8:32:53:80:2c:34:94:69:c7:0f:69:d7:1b:85:
    8b:7a:13:32:b3:57:20:2e:12:b4:4a:85:97:69:91:1c:a6:9c:
    99:84:66:5e:51:dc:e6:a1:60:6f
```

Fig. 8 Example Shibboleth proxy credential

Fig. 9 Extraction of the user role from the Shibboleth proxy credential

```
Input: Shibboleth proxy credential of a user
Output: User role for accessing a resource
Method:
    Get the delegated credential from the client
    foreach certificate in the credential
      Get the extension of the certificate
       if non-critical extension exists then get the SAML assertions

      foreach SAML assertion
          parse the SAML assertion and obtain the attribute name, value and the namespace
      foreach attribute
            if the attribute name refers to the user role
              then return the attribute value representing the role name
            else return the identity of the user
```

non-critical extension is 1.3.6.1.4.1.3536.1.1.1.10. So, if a non-critical extension with that OID is present, then the server obtains the SAML assertions present in it. From the SAML assertions, the server obtains the attributes of the user, such as the user's role name, and uses them to perform the role-mapping (from a VO role to a local role) and all the authorization decisions. If the server could not obtain the SAML assertions from the credential, then it performs the matching between the user's distinguished name (DN) present in the certificate and the DN stored in the MCAT database.

Our server-side implementation extends the San Diego Supercomputer Center (SDSC) Authentication/Integrity of Data (AID) library which integrates the Grid Security Infrastructure (GSI) into the SRB system. The AID library provides an API to the underlying Generic Security Services API (GSSAPI) package, shielding the application programmer from the complexities of the GSSAPI. It also supports Kerberos and the Distributed Computing Environment (DCE) security via GSSAPI.

5 Performance Analysis

We implemented our proposed RBAC system and analyzed its performance in terms of the overheads incurred. Our implementation is compared with the existing implementation of SRB, which does not use Shibboleth for access control. For our implementation, we installed an MCAT-enabled SRB (MES) server on a SuSE Linux machine with a 2.6 GHz Intel Pentium IV processor and 1 GB RAM. SRB version 3.4.2 is used, and Oracle 9.0 is used to manage the MCAT database. To use the GSI, we installed Globus Toolkit 4.0.2. To serve the user requests, a non-MES server is installed on another SuSE Linux machine with a 1.6 GHz Intel dual-core processor and 2 GB RAM. Shibboleth identity provider (IdP) 1.3c was configured to run on the SSL-enabled Apache 2.2.0 and Tomcat 5.0.28 servers. We used OMAR 3.0 beta 1 as the repository for storing the XACML policies. Our MES and non-MES servers are connected to a Fast Ethernet LAN. For the purpose of analysis, the *Sput* command was invoked on the non-MES server, which is an SRB command line utility for importing a local file/directory at a site into the SRB system.

5.1 Profiling Details

For profiling the time taken by the Shibboleth interface program (SIP), a Java method *System.currentTimeMillis*() was used to get the time in milliseconds. Also, for the SRB system analysis, a C time function *gettimeofday*() was used to get the time in microseconds. For more accuracy, the servers were shutdown and restarted before each client invocation of the SIP and the *Sput* command, in order to minimize the caching effects.

The calls made to the original implementation of the SRB server using the user's proxy credential (referred to as *Original* in the figures) and the modified implementation of the SRB server using the user's Shibboleth proxy credential (referred to as *Modified* in the figures), respectively, are profiled. In case of using the Shibboleth proxy credential, an overhead for invoking the SIP is incurred. The SIP obtains the attributes of the user, such as a VO role, in the form of an assertion from

the Shibboleth, and then embeds the assertion into the user's proxy credential. The total time for this process is approximately 6 seconds, and the default life time of the user's proxy credential is 12 hours. However, this overhead is incurred only once before the requests are made by the user to the SRB server, and the user can make any number of requests before the proxy credential expires.

The *Sput* command was executed to transfer the data files of different sizes from our non-MES server to the MES server. Figure 10 shows the total times taken for 100 KB, 10 MB and 1 GB data files in the original and modified SRB implementations. We can see that our RBAC system adds a very small overhead, compared to the total execution time required by the original SRB implementation. And the overhead is independent of the data file size. To analyze the overhead further, we profiled the security functions on the client-side and server-side of the SRB system, and the results are described in the following sections.

5.2 Client-Side Security

To analyze the overhead at the client side, the *Sput* command for transferring a 100 KB data file is used, and the execution times of the following functions of the AID library are measured:

1. The *aid_setup_creds* function sets up the user's proxy credential on the client-side.
2. The *aid_establish_context_clientside* function performs the mutual authentication on the client-side.

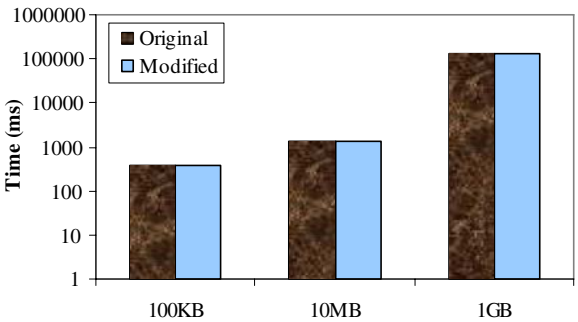

Fig. 10 Total execution time for the *Sput* command

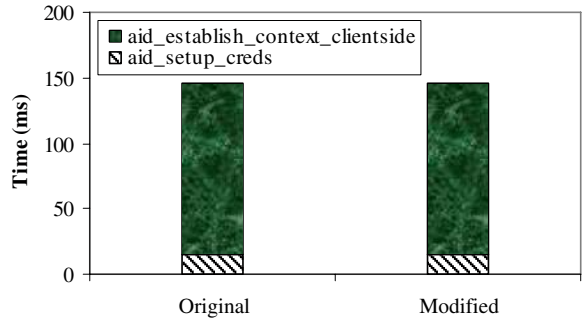

Fig. 11 Security overheads on the client-side

The authentication mode is set to GSI delegation, and as shown in Fig. 11, the time for setting up the credentials and the time for establishing the context for mutual authentication are almost the same in both implementations. The reason is because all the security functions on the client-side remain unchanged in our modified implementation, except for using the Shibboleth proxy credential instead of the user proxy credential.

5.3 Server-Side Security

On the server-side, the authentication mode is also set to GSI delegation, and we measured the execution times of the following functions:

1. The *aid_setup_creds* function sets up the server's credential.
2. The *aid_establish_context_serverside_with_delegate* performs the mutual authentication on the server-side and extracts the user's VO role or DN from the proxy credential delegated to the server.

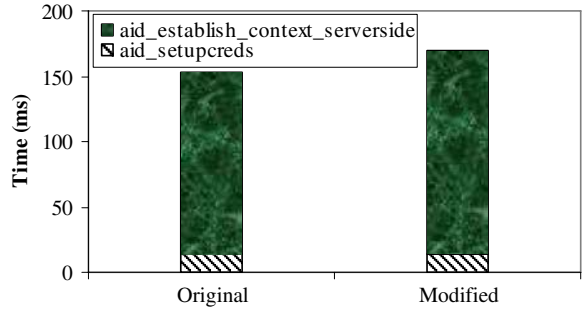

Fig. 12 Security overheads on the server-side

As shown in Fig. 12, the execution time of the *aid_setup_creds* function remains the same because there is no change in the server credential. The *aid_establish_context_serverside_with_delegate* function takes a little bit more time in the modified implementation, because of the additional time for parsing the user attributes in the Shibboleth proxy credential and extracting the VO role. However, the time difference is in the order of a few milliseconds, which is negligible compared to the total time taken for a typical SRB application, like the *Sput* operation.

6 Conclusion

In this paper, we enhanced the access control mechanism of SRB by using Shibboleth, XACML, and OMAR to support role-based access control (RBAC) and the distributed administration of the access control policies. The proposed RBAC system allows quick and easy deployments, and provides privacy protection for the users. Furthermore, the users can be dynamically granted memberships on the VO roles. The administration overhead of the resource providers is reduced because the information about the VO roles and their mapping to local roles is maintained in the MCAT database, so the resource providers do not need to maintain the information about individual users and their access privileges. The specification of access control policies at the VO level eliminates unnecessary authentication, mapping and connections by denying invalid requests at the VO level.

As our RBAC system uses the push model, the SRB server does not need to contact the attribute service to obtain the user information for each access request. So it is more efficient in terms of performance when we have a large number of user requests. In Grids, as users and resources are dynamic, administrative scalability is also critical. As users join/leave the organization, the administrator may have to update the policies as well as the necessary permissions of individual users. Considering the frequency of these changes, our RBAC system has good administrative scalability because the administrator just needs to grant/revoke the role memberships of the users when they join/leave the system.

Our performance analysis shows that the proposed RBAC system incurs a small overhead for authorizing the user. However, this overhead is quite acceptable when we consider the benefits of our system, such as the scalability in terms of the number of users, reduced administration overhead of the resource providers, and the distributed administration of the access control policies.

References

1. Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Gianoli, A., Spataro, F., et al.: Managing dynamic user communities in a Grid of autonomous resources. In: Proceedings of International Conference for Computing in High Energy and Nuclear Physics. La Jolla, California (2003)
2. Antonioletti, M., Atkinson, M., Baxter, R., Borley, A., Chue Hong, N., Dantressangle, P., Hume, A., et al.: OGSA-DAI status and benchmarks. In: Proceedings of the UK e-Science All Hands Meeting. Nottingham, UK (2005)
3. Atkinson, M., Karasavvas, K., Antonioletti, M., Baxter, R., Borley, A., Chue Hong, N., Hume, A., et al.: A new architecture for OGSA-DAI. In: Proceedings of the UK e-Science All Hands Meeting. Nottingham, UK (2005)
4. Baker, M., Apon, A., Ferner, C., Brown, J.: Emerging Grid standards. Computer **38**(4), 43–50 (2005)
5. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The SDSC storage resource broker. In: Proceedings of Conference of the Centre for Advanced Studies on Collaborative Research. Toronto, Ontario, Canada (1998)
6. Baru, C., Rajasekar, A.: A hierarchical access control scheme for digital libraries. In: Proceedings of the 3rd ACM Conference on Digital Libraries, pp. 275–276. Pittsburgh, PA (1998)
7. Butler, R., Welch, V., Engert, D., Foster, I., Tuecke, S., Volmer, J., Kesselman, C.: A national-scale authentication infrastructure. Computer **33**(12), 60–66 (2000)
8. Carmody, S.: Shibboleth overview and requirements. Shibboleth Working Group Document, Available via http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-requirements-01.html (2001)
9. Demchenko, Y., de Laat, C., Gommans, L., van Buuren, R.: Domain based access control model for distributed collaborative applications. In: Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (2006)
10. Ferraiolo, D., Kuhn, R.: Role-based access control. In: Proceedings of the 15th National Computer Security Conference. Baltimore, MD (1992)
11. Ferraiolo, D., Barkley, J., Kuhn, D.R.: A role-based access control model and reference implementation

within a corporate intranet. ACM Trans. Inf. Syst. Secur. **2**(1), 34–64 (1999)
12. Foster, I., Kesselman, C.: Security, accounting, and assurance. In: Foster, I., Kesselman, C. (eds.) The Grid: Blueprint for a New Computing Infrastructure, pp. 395–420. Morgan Kaufmann, San Francisco (1999)
13. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: enabling scalable virtual organizations. Int. J. Supercomput. Appl. High Perform. Comput. **15**(3), 200–222 (2001)
14. Foster, I., Grossman, R.L.: Data integration in a bandwidth-rich world. Commun. ACM **46**(11), 50–57 (2003)
15. Freudenthal, E., Pesin, T., Port, L., Keenan, E., Karamcheti, V.: dRBAC: distributed role-based access control for dynamic coalition environments. In: Proceedings of the 22nd International Conference on Distributed Computing Systems, pp. 411–420. Vienna, Austria (2002)
16. The Globus Security Team: Globus Toolkit version 4 Grid security infrastructure: a standards perspective. Available via http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf (2005)
17. Humphrey, M., Thompson, M.R., Jackson, K.R.: Security for Grids. Proc. IEEE **93**(3), 644–652 (2005)
18. Joshi, J.B.D., Bhatti, R., Bertino, E., Ghafoor, A.: Access-control language for multidomain environments. IEEE Internet Comput. **8**(6), 40–50 (2004)
19. Joshi, J.B.D., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng. **17**(1), 4–23 (2005)
20. Lee, H.K., Luedemann, H.: A lightweight decentralized authorization model for inter-domain collaborations. In: Proceedings of the ACM Workshop on Secure Web Services, pp. 83–89. Fairfax, VA (2007)
21. Lorch, M., Proctor, S., Lepro, R., Kafura, D., Shah, S.: First experiences using XACML for access control in distributed systems. In: Proceedings of the ACM Workshop on XML Security, pp. 25–37 (2003)
22. Mayfield, T., Roskos, J.E., Welke, S.R., Boone, J.M.: Integrity in automated information systems. Technical Report, National Computer Security Center (1991)
23. MCAT. Available via http://www.sdsc.edu/srb/index.php/MCAT
24. Nagaratnam, N., Janson, P., Dayka, J., Nadalin, A., Siebenlist, F., Welch, V., Foster, I., Tuecke, S.: The security architecture for open Grid services. Open Grid Service Architecture Security Working Group, Global Grid Forum (2002)
25. Organization for the Advancement of Structured Information Standards (OASIS): ebXML registry technical committee. Available via http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=regrep
26. Organization for the Advancement of Structured Information Standards (OASIS): Assertions and protocols for the OASIS Security Assertion Markup Language (SAML) v1.1. Available via http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security (2003)
27. Organization for the Advancement of Structured Information Standards (OASIS): Core and hierarchical role based access control (RBAC) profile of XACML v2.0. Available via http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf (2005)
28. Organization for the Advancement of Structured Information Standards (OASIS): eXtensible Access Control Markup Language (XACML) version 2.0. Available via http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf (2005)
29. Organization for the Advancement of Structured Information Standards (OASIS): ebXML registry information model version 3.0. Available via http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rim-3.0-os.pdf (2005)
30. Organization for the Advancement of Structured Information Standards (OASIS): SAML 2.0 profile of XACML v2.0. Available via http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf (2005)
31. Object, Metadata and Artifacts Registry. Available via http://ebxmlrr.sourceforge.net/3.0/
32. Otenko, S., Chadwick, D.: A comparison of the Akenti and PERMIS authorization infrastructures. Available via http://sec.isi.salford.ac.uk/download/AkentiPERMISDeskComparison2–1.pdf (2003)
33. Pearlman, L., Welch, V., Foster, I., Kesselman, C., Tuecke, S.: A community authorization service for group collaboration. In: Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks. Monterey, CA (2002)
34. Pereira, A.L., Muppavarapu, V., Chung, S.M.: Role-based access control for Grid database services using the community authorization service. IEEE TDSC **3**(2), 156–166 (2006)
35. Pereira, A.L., Muppavarapu, V., Chung, S.M.: Managing role-based access control policies for Grid databases in OGSA-DAI using CAS. J. Grid Comput. **5**(1), 65–81 (2007)
36. Rajasekar, A., Wan, M., Moore, R.: MySRB & SRB: components of a data Grid. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, pp. 301–310. Edinburgh, Scotland, UK (2002)
37. Rajasekar, A., Wan, M., Moore, R., et al.: Storage resource broker-managing distributed data in a Grid. Comput. Soc. India J. **33**(4) (2003)
38. Ramaswamy, C., Sandhu, R.S.: Role-based access control features in commercial database management systems. In: Proceedings of the 21st National Information Systems Security Conference. Arlington, VA (1998)
39. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)
40. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM. Trans. Inf. Syst. Secur. **2**(1), 105–135 (1999)
41. Sandhu, R., Ferraiolo, D.F., Kuhn, D.R.: The NIST model for role based access control: towards a unified standard. In: Proceedings of the 5th ACM

Workshop on Role Based Access Control. Berlin, Germany (2000)
42. Scavo, T., Welch, V.: A Grid authorization model for science gateways. In: International Workshop on Grid Computing Environments. Reno, NV (2007)
43. Secretariat of Information Technology Industry Council (ITI): American National Standard for Information Technology—Role based access control. Available via http://csrc.nist.gov/rbac/rbac-std-ncits.pdf (2003)
44. Thompson, M.R., Essiari, A., Keahey, K., Welch, V., Lang, S., Liu, B.: Fine-grained authorization for job and resource management using Akenti and the Globus Toolkit. In: Proceedings of International Conference for Computing in High Energy and Nuclear Physics. La Jolla, California (2003)
45. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., Tuecke, S.: Security for Grid services. In: Proceedings of the 12th International Symposium on High-Performance Distributed Computing, pp. 48–57. Seattle, WA (2003)
46. Welch, V., Barton, T., Keahey, K., Siebenlist, F.: Attributes, anonymity, and access: Shibboleth and Globus integration to facilitate Grid collaboration. In: Proceedings of the 4th Annual PKI R&D Workshop. Gaithersburg, MD (2005)