J Grid Computing (2007) 5:65–81
DOI 10.1007/s10723-006-9054-4

Managing Role-Based Access Control Policies for Grid Databases in OGSA-DAI Using CAS

Anil L. Pereira · Vineela Muppavarapu · Soon M. Chung

Received: 9 January 2006 / Accepted: 12 October 2006 / Published online: 28 December 2006

Abstract In this paper, we present a role-based access control method for accessing databases through the Open Grid Services Architecture – Data Access and Integration (OGSA-DAI) framework. OGSA-DAI is an efficient Grid-enabled middleware implementation of interfaces and services to access and control data sources and sinks. However, in OGSA-DAI, access control causes substantial administration overhead for resource providers in virtual organizations (VOs) because each of them has to manage a role-map file containing authorization information for individual Grid users. To solve this problem, we used the Community Authorization Service (CAS) provided by the Globus Toolkit to support the role-based access control (RBAC) within OGSA-DAI. CAS uses the Security Assertion Markup Language (SAML). Our method shows that CAS can support a wide range of security policies using role-privileges, role hierarchies, and constraints. The resource providers need to maintain only the mapping information from VO roles to local database roles and the local policies in the role-map files, so that the number of entries in the role-map file is reduced dramatically. Also, unnecessary authentication, mapping and connections can be avoided by denying invalid requests at the VO level. Thus, our access control method provides increased manageability for a large number of users and reduces day-to-day administration tasks of the resource providers, while they maintain the ultimate authority over their resources. Performance analysis shows that our method adds very little overhead to the existing security infrastructure of OGSA-DAI.

Key words Open Grid Services Architecture – Data Access and Integration (OGSA-DAI) · Grid databases · virtual organization (VO) · Community Authorization Service (CAS) · role-based access control (RBAC)

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

1 Introduction

Grid has emerged recently as an integration infrastructure for sharing and coordinated use of diverse resources in dynamic, distributed virtual organizations (VOs) [1–6]. The use of databases in Grids presents different security needs and access policies compared with the use of computational resources. For example, certain applications may be authorized to access only a certain part of a database during a specific time interval.

The Data Access and Integration Services Working Group (DAIS-WG) of the Global Grid Forum (GGF) is currently establishing the standards for Grid interface to data resources [7]. The Open Grid

Services Architecture – Data Access and Integration (OGSA-DAI) provides the first implementation for these emerging standards. Currently, OGSA-DAI supports role-based access control (RBAC) [8, 9] via a role-map file that maps individual Grid users to database roles. In other words, permissions are associated with roles, and users are made members of appropriate roles, thereby acquiring the roles' permissions [10]. In this case, each resource provider has to maintain a role-map file to authorize access to its resources. This access control method is not suitable for VOs, because both users and resources are dynamic in VOs. Multiple entries in multiple role-map files may need to be updated if new users are allowed to access multiple data resources or if the access privileges of current users change. This puts an unnecessary burden on the resource providers in managing the role-map files, especially when both the users and resource providers belong to multiple VOs. Furthermore, there are unnecessary overheads on the resource providers whenever users make invalid requests. This is because users are authenticated, mapped and connected to the databases without first verifying their requests against their access privileges.

In this paper, we describe how the Community Authorization Service (CAS) provided by the Globus Toolkit [11] can be used to enhance the security mechanism in OGSA-DAI. The CAS records user groups and their permissions on resources, and it targets access control for computational and file-based storage resources. But, we demonstrate that the CAS can also support RBAC for multiple VOs to access Grid databases within the OGSA-DAI framework. We extended the RBAC approach supported by OGSA-DAI to allow users to be assigned memberships on VO roles, to assign privileges and specify constraints on those roles, and to allow role hierarchies.

In our method, CAS maintains the security policies of VOs, grants users' memberships on VO roles, and then authorizes them in those roles. The resource providers need to maintain only the mapping information from VO roles to local database roles and the local policies, thus the number of entries in the role-map file is reduced dramatically. Our method 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 itself. This eliminates unnecessary authentication, mapping and connection overheads on the resource providers. When users join/leave a VO, the resource providers do not have to bother about individually adding/removing their information in the role-map files because the CAS server can just grant/revoke their memberships on the VO roles. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately in the role-map files. This enables the resource providers to have the ultimate authority over their resources.

We have implemented our proposed method and analyzed its performance. In our implementation, users obtain CAS credentials based on user credentials. The user credential is formed by an X.509 certificate and the associated public/private keys, and it is issued by a Certificate Authority (CA) trusted by all entities in a Grid [12]. The CAS credentials contain the authorization information for the user in terms of his/her VO roles. We have extended the client-side implementation of OGSA-DAI to pass the CAS credential. The server-side has been extended to parse the CAS credential to obtain the VO roles. The server also verifies the capabilities associated with that VO role against the local policies of the resource provider and maps it to a local database role. We have evaluated our solution in terms of the overheads incurred when security contexts are set up between a client and a server. This has been done with respect to the original security mechanism in OGSA-DAI.

The organization of the paper is as follows: Section 2 explains the current authorization mechanism in OGSA-DAI. In Section 3, we describe CAS. In Section 4, we present our RBAC method using CAS in OGSA-DAI. Section 5 describes the implementation details, and Section 6 describes the results of performance analysis. Section 7 contains conclusions.

2 OGSA-DAI

OGSA-DAI is an efficient Grid-enabled middleware implementation of interfaces and services to access and control data sources and sinks [13]. In order to expose physical data resources to the Grid, by extending the interfaces defined by Open Grid Services Infrastructure (OGSI) [14], OGSA-DAI introduced the following services [13]: (1) Grid Data Service Factory (GDSF): represents a data resource, and exposes its capabilities and metadata. (2) Grid

Data Service (GDS): created by a GDSF and holds the client session with the data resource. (3) DAI Service Group Registry (DAISGR): clients can discover service/data by locating GDSFs registered with a DAISGR.

Figure 1 shows how clients can access data resources using OGSA-DAI. The client first contacts the DAISGR and gets information about the registered GDSFs. The client then contacts the desired GDSF and makes a request for the creation of a GDS. Once the GDS is created, it authorizes the client and establishes a JDBC connection to the underlying database. The client can then submit queries on the database and retrieve results. The client authorization process is discussed in detail in Section 2.1.

2.1 RBAC and Current Authorization Mechanism in OGSA-DAI

User authorization is one of the most challenging issues in Grid computing. Current authorization mechanisms cannot address all the issues that arise in dynamic Grid environments which often encompass multiple organizations, each with its own security policy [15]. RBAC shows clear advantages over traditional discretionary and mandatory access control models in such environments, because it allows the uniform representation of diverse security policies and ensures that no security violations occur during inter-domain access [15].

Furthermore, RBAC is distinguished by its inherent support for the Principle of Least Privilege [16]. The Principle of Least Privilege requires that a user be given no more privileges than necessary to perform a job [8]. 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. Hence, some Grid authorization mechanisms have adopted the RBAC model. With our method, users can request a particular role among those they are entitled to and, hence, gain the specific permissions tied with that role.

The current security infrastructure of OGSA-DAI uses a role-map file for authorizing a Grid user's request. The role-map file contains the information for mapping a Grid user credential to a username and a password that are used to connect to a database at a particular authorization level. Multiple entries in multiple role-map files may need to be updated if new Grid users are allowed to access multiple data resources or if the access privileges of current users change. Thus, managing the entries in a role-map file is difficult. With these considerations in mind, we propose an efficient access control mechanism for Grid database services in OGSA-DAI.

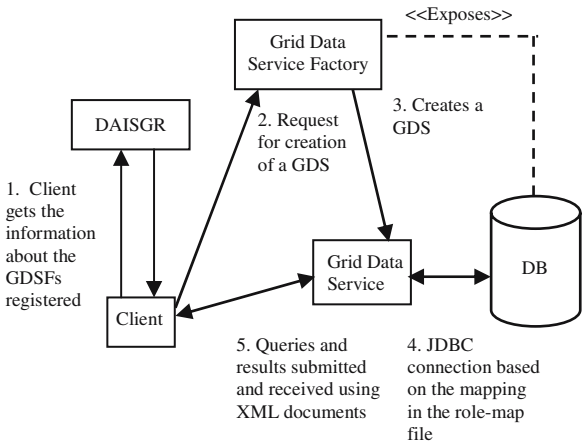

Fig. 1 Accessing a data resource through OGSA-DAI

3 Community Authorization Service (CAS)

We enhanced the existing implementation of OGSA-DAI to use the Community Authorization Service (CAS) provided in the Globus Toolkit. CAS provides a scalable mechanism for specifying and enforcing complex and dynamic policies that govern resource usage within Grids. It allows resource providers to delegate some of the authority for maintaining fine-grain access control policies to communities, while still maintaining the ultimate authority over their resources [17].

The following is the sequential process of a Grid user obtaining a CAS credential for accessing a resource. As shown in Fig. 2, a user generates a certificate (Cu) by making a request to a Certificate Authority (CA) which is trusted by all the entities within the Grid, i.e., all users and resources. If a user needs to gain access to a resource, the user generates a proxy credential (Cup) which is signed by his/her user certificate (Cu). This generated proxy credential's lifetime will be less than the lifetime of the user certificate. The lifetime of a proxy credential generated using the Globus Toolkit is 12 hours. In order to use a CAS credential, the user makes a request to the

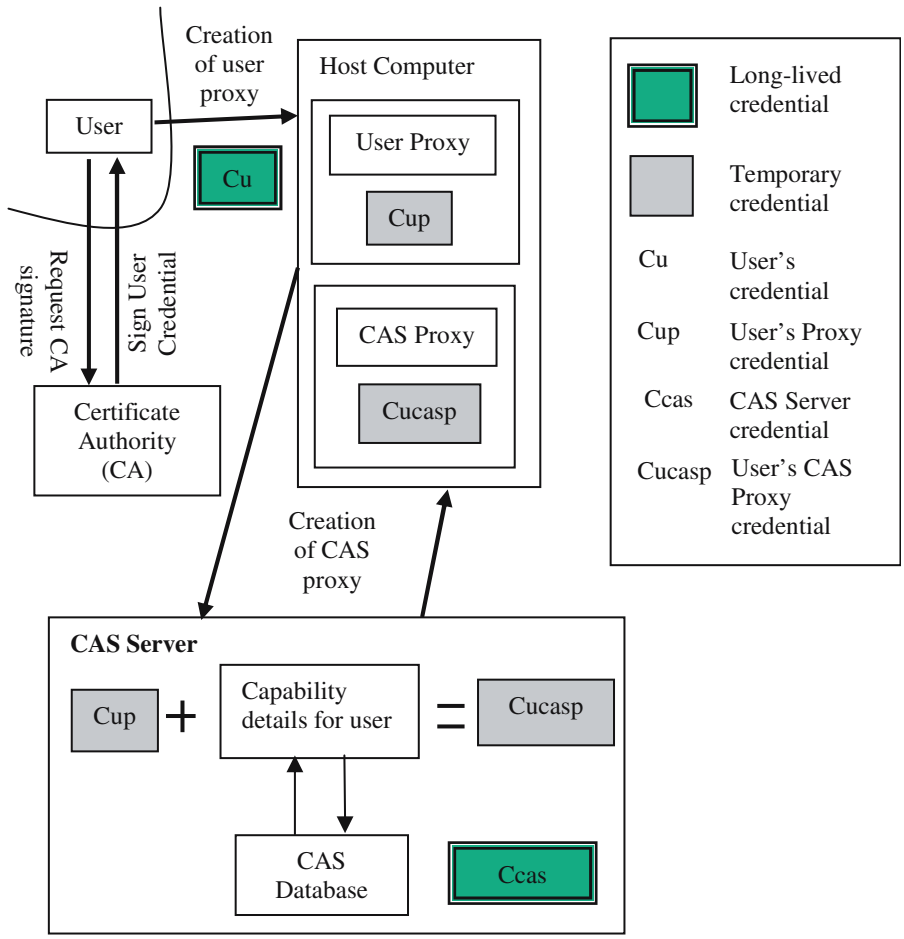

Fig. 2 User's normal proxy credential and CAS proxy credential creation

CAS server to initiate a CAS proxy based on the user's proxy credential. The CAS server authenticates the user and obtains the user's capability details present in the CAS database. The CAS server then creates a CAS proxy credential (Cucasp) which contains the CAS policy assertions to represent the user's capabilities and restrictions as an extension to the existing user proxy credential (Cup). The CAS proxy credential is presented to the resource provider. The resource provider verifies the validity of the CAS proxy credential and then parses the CAS policy assertions to obtain the restrictions imposed by the CAS server. Thus, the CAS credential facilitates the mapping of the user to a local user account, and the capabilities and restrictions determine the operations the user is allowed to perform.

Due to the CAS system design and RBAC, our method provides scalability in terms of the number of users and VOs. CAS reduces the number of necessary trust relationships from $C \times P$ to $C+P$, when there are C consumers and P providers. Each consumer needs to be known and trusted by the CAS server, but not by each provider. Similarly, each provider needs to be known and trusted by the CAS server, but not by each consumer [17]. A single CAS server can support the authorization for multiple VOs. Also, it has been shown that the cost of administering RBAC is proportional to $U+P$ per role, while the cost of associating users directly with permissions is proportional to $U \times P$, where U is the number of individuals in a role and P is the number of permissions required by the role [19, 20].

However, in terms of the actual number of access requests on resources, using a single CAS server may not be quite scalable. 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. A possible solution for these problems

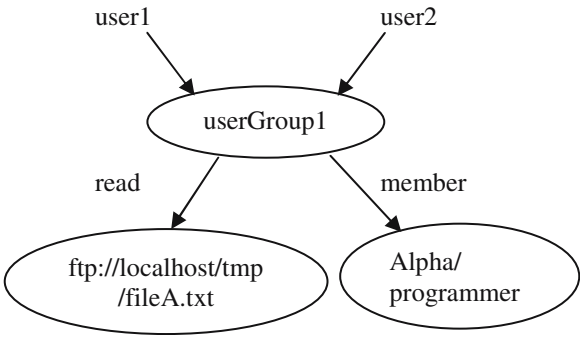

Fig. 3 userGroup1 with a role Alpha/programmer and read access to ftp://localhost/tmp/fileA.txt

depends on how frequently the community policies change. If the community policies do not change frequently, a single master server can be maintained to accept the changes and then routinely replicate the policies to one or more read-only slave servers. If the community policies change frequently, multiple peer servers can be used. All the servers update the policies, so that the failure of any one server will not lead to a loss of functionality [17]. However, when policies are changing dynamically, the complete centralization of policies can achieve better consistency. Also, in the case that a user credential is compromised, revocation is easier when a single CAS server is used because the user needs to be removed only from that server [17].

4 RBAC with CAS in OGSA-DAI

In a VO with a large number of users, we could think of several groups of users, each with different levels of access (roles). A role has certain privileges associated with it. When a VO role is mapped to a local role, it will acquire the access rights associated with the local role, such as the right to perform the database operation SELECT on specific tables of a database during certain time intervals. For mapping, the resource provider can obtain the VO role through the user's CAS credential. A VO role can be assigned to any number of users. When users join/leave the VO, the resource providers do not have to bother about individually adding/removing them from the role-map files because the CAS server could just grant/revoke their membership from the existing VO roles. Moreover, if roles and privileges do not change often, the resource provider does not need to update its role-map file frequently.

4.1 Drawbacks of the Existing Approach for RBAC with CAS

A proposed approach for supporting RBAC with CAS is the use of rights associated with a role to access role-specific resources [21]. The role of a user is presented in a hierarchical form. For example, Alpha/admin indicates the administrator role of a virtual organization Alpha. Alpha could be the name of a project undertaken by collaborating organizations. In VOs, users may be assigned specific tasks, and there may be constraints related to the execution of those tasks. For example, a user may have access to data only during certain days of the week. One of the key aspects of RBAC is that it allows the specification of constraints on roles [9]. However, the approach proposed in [21] does not address this aspect of RBAC. Another drawback is that for a user to act in multiple roles, multiple CAS proxy certificates have to be created.

Most systems do not enforce the Principle of Least Privilege [24]. An application must be delegated only those privileges required for completing a certain set of tasks, otherwise the application should be totally trusted to do no more than required. In Grids, this is even more critical since software can be regularly downloaded from remote sites. In addition to the possibility of downloading malicious software such as Viruses, Trojans, Worms, and so on, we cannot expect software to work exactly as specified because of bugs or malicious intent. Any software with certain extra privileges has the potential to cause severe damage to computer systems and data. When the method proposed in [21] is used in CAS, the Principle of Least Privilege is not always enforced. Users authorized to act in a role may be granted some privileges

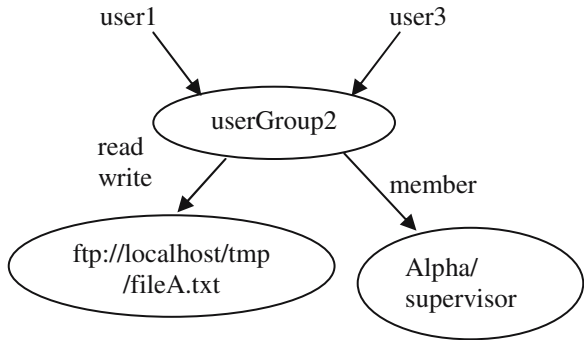

Fig. 4 userGroup2 with a role Alpha/supervisor and read/write access to ftp://localhost/tmp/fileA.txt

in addition to those assigned to that role. This is because both roles and privileges are set up in the same way. In particular, a role is considered as a resource, and a user group is given the "member" right on a role, in the same way that a user group is given the "read" right on a resource such as a file. With the current implementation of CAS, a user belonging to multiple groups can request and be authorized any combination of roles and privileges from one or more of those groups at the same time.

The following example illustrates this problem. The VO Alpha may have a policy in which programmers are allowed only read access to a particular file while supervisors are allowed read/write access. To implement this policy based on the method proposed in [21], two roles, "Alpha/programmer" and "Alpha/supervisor," can be created as shown in Figs. 3 and 4. The "Alpha/programmer" role can be assigned the "read" right on "ftp://localhost/tmp/fileA.txt," and the "Alpha/supervisor" role can be assigned the "read" and "write" rights on "ftp://localhost/tmp/fileA.txt." As users of userGroup1 are given the "member" right on the role "Alpha/programmer," they can acquire the "read" right on "ftp://localhost/tmp/fileA.txt." Similarly, as users of userGroup2 are given the "member" right on the role "Alpha/supervisor," they can acquire the "read" and "write" rights on "ftp://localhost/tmp/fileA.txt." If a user "user1" is in both userGroup1 and userGroup2, and makes a request to act in the "Alpha/programmer" role with the "read" and "write" rights on "ftp://localhost/tmp/fileA.txt," CAS will authorize the request because "user1" is a member of both user groups. This authorization decision clearly violates the VO policy in terms of the Principle of Least Privilege, since a programmer is granted write access to a file while he/she is allowed only read access to it. If there is an application that analyzes data for programmers, it must be delegated only the read access to the file. Delegating the write access to the application can potentially result in an alteration of the file.

A possible refinement to this method is distributing the VO policies to the resource providers while keeping only the assignment of users to the VO roles within CAS. This method is applicable if roles and privileges do not change often and VOs have a long lifespan. CAS is not used to associate the privileges with roles to access role-specific resources. Instead, the VO role is mapped to a local role, and the assignment of fine-grain privileges to the local role is the responsibility of the resource provider. The fine-grain privileges associated with the local role can be negotiated between the VO and the resource provider.

A user can delegate a subset of his/her authorized VO roles to certain applications and services. In this case, the privileges associated with the delegated VO roles are the privileges associated with the corresponding local roles. As shown in Fig. 5, after a negotiation with a VO, a resource provider could decide to map the "Alpha/supervisor" role to a local "Supervisor" role that allows the function updateInventory( ) to be performed on a database (DB_1). Another resource provider could decide to map the same VO role to a local "Employee" role that allows the function viewInventory( ) to be performed on another database (DB_2). This method enforces the

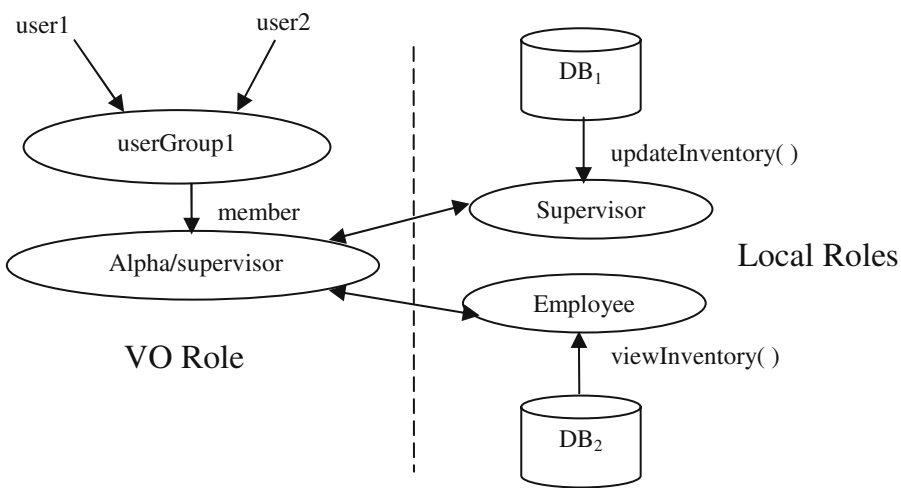

Fig. 5 Distributing VO policies to the resource providers

Principle of Least Privilege since a user can receive no more privileges for a VO role other than those tied with the corresponding local roles. However, a VO does not have the flexibility to update its policy without contacting the resource providers because they control the assignment of privileges to the local roles.

4.2 Our Proposed Method for RBAC Using CAS

Specification of policies at the VO level allows authorization decisions to be made based on the user's request and VO policies. In case the user does not possess the required privileges, the access can be denied at the VO level itself without involving the resource providers. This eliminates authentication, mapping and connection overheads on the resource providers in case the request is not valid.

Our proposed method is implemented using a newer version of CAS which supports SAML [18]. Participating organizations within VOs may have different security models. So, it is important for these models to interoperate at different levels of trust, and SAML can be used to uniformly express the authorization assertions between different security domains.

The CAS server contains policy statements that specify who (which user or group) has the permission, which resource or resource group the permission is granted on, and what permission is granted [17]. The permission is denoted by a *service type* and an *action*. The *action* describes the operation (e.g., read, write or execute program), and the *service type* defines the namespace in which the *action* is defined (e.g., file). Different resource providers may recognize different service types, but all resource providers that recognize the same *service type* should have the same interpretation of that service type's actions [17].

To support RBAC using CAS, we define the role as a new *service type*, and each role name in the form of "[VOName{,SubgroupName}][,RN=rolename]"[1] as an *action*. Roles can be specified for any subgroup within a VO. For example, "Alpha,RN=Manager" indicates the Manager role for the Alpha VO, whereas "Alpha,Data,RN=Manager" indicates the Manager role for the Data subgroup of the Alpha VO.

[1] The curly brackets {} indicate zero or more occurrences of their content and the square brackets [] indicate only one occurrence of their content.

For each role name, we can specify the actions (privileges and constraints) and some junior roles. Resources represented in the form, "URI{.Subcomponent}" are associated with usergroups in the CAS database. Thus, fine-grain authorization for resources can be allowed, where access control can be specified not only for the entire resource (e.g. database) but also for the subcomponents of a resource (e.g. table). For example, "http://130.108.17.176:8080/ogsa/services/ogsadai/SecureGridDataServiceFactory.Employee" indicates the Employee table in the database represented by the specified URI. This permits the members of a usergroup to access a resource in a specific role. We propose new service types and actions to assign privileges on roles, and to specify timing constraints as described in the following subsections. With these proposed ideas, privileges can be specified at fine-grain levels.

4.2.1 Specifying Privileges and Timing Constraints on VO Roles

To assign privileges on a role, we define the role name as a *service type* and each privilege in the form of "privilege:operation" as an *action*. This allows the specification of a privilege in terms of the operation permitted for a specific role. For example, the role name "Alpha,RN=Manager" could have "privilege:select" as a privilege to execute the SELECT operation. Obviously, not only the basic database operations, but also complex operations, such as transactions and stored procedures, can be assigned as privileges.

To specify a timing constraint on a role, we define the role name as a *service type* and the timing constraint in the following form as an *action*:

"timing_constraint:[local/GMT] [Date#Day#Time]{;Date#Day#Time}" where

- Date can be "[FromDate-ToDate]{,FromDate-ToDate}"
- Day can be "[FromDay-ToDay]{,FromDay-ToDay}" or "[Day]{,Day}"
- Time can be "[FromTime-ToTime]{,FromTime-ToTime}"

For example, the role name "Alpha,RN=Manager" could have "timing_constraint:GMT#10.01.2005–07.30.2006#Mon–Fri#1:00–5:00,17:00–21:00," indicating that the user can act in that role only within the time intervals 1:00–5:00 and 17:00–21:00 GMT from Monday to Friday during 10.01.2005–07.30.2006.

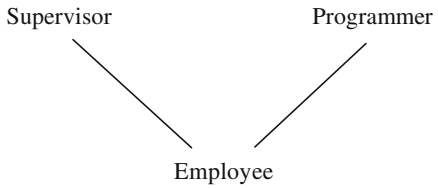

Fig. 6 An example of limited role hierarchy

4.2.2 Specifying Role Hierarchies

A role hierarchy defines a seniority relation between roles, whereby senior roles automatically acquire the permissions of the junior roles. In the role hierarchy diagrams [22], senior roles are placed at the top of the junior roles.

According to the NIST standard for RBAC [22], there are two types of role hierarchies: limited hierarchy and general hierarchy. In the limited hierarchy, each senior role cannot have more than one junior role. On the other hand, in the general hierarchy, each senior role can have multiple junior roles. However, in both types, a junior role can have multiple senior roles. Examples of a limited hierarchy and a general hierarchy are illustrated in Figs. 6 and 7, respectively.

The selection of the type of role hierarchy is made by the VO. To specify a role hierarchy, we define each senior role name as a *service type* and each junior role name in the form of "junior_role: [VOName{,SubgroupName}][,RN=rolename]" as an *action*. For example, in Fig. 7, the senior role name "Alpha,RN=Manager" has "junior_role:Alpha,RN=Supervisor" and "junior_role:Alpha,RN=Programmer" as junior role names, and thereby inherits their privileges.

The constraints on a junior role are also inherited by a senior role [22]. In our method, the timing constraint specified on a senior role would override those on the junior roles. If a timing constraint is not specified on a senior role, then it inherits the timing constraints of its junior roles. However, there should be no conflicts between the timing constraints on the junior roles. If such conflicts exist, then the concept of limited inheritance [9] can be used. With limited inheritance, a senior role can inherit only a subset of privileges of a junior role.

The following example illustrates the concept of limited inheritance. As shown in the hierarchy of Fig. 7, the Manager role is senior to both the Supervisor and Programmer roles. Managers can be prevented from inheriting specific privileges of the Supervisor role by defining a new role Supervisor' as shown in Fig. 8. Only those specific privileges not to be inherited by the Manager role can be assigned to the Supervisor' role and the rest of the original set of privileges can be retained by the Supervisor role. The Supervisor' role can inherit the privileges from the Supervisor role, thus acquires the entire set of privileges originally held by the Supervisor role. The Manager role can then inherit the privileges of the Supervisor role but not the privileges of the Supervisor' role. Similarly, by creating the Programmer' role, the Manager role can be prevented from inheriting specific privileges originally held by the Programmer role.

To deal with conflicting constraints on junior roles, some modifications to the approach illustrated above are required. If conflicting timing constraints exist on the Supervisor and Programmer roles shown in Fig. 7, then new timing constraints with no conflicts can be specified on those roles while each retains the entire set of its privileges. The Manager role can then inherit the privileges and new constraints on the Supervisor and Programmer roles. The original timing constraints of the Supervisor and Programmer roles can be specified on the Supervisor' and Programmer' roles, respectively. These timing constraints will then override the new constraints specified on the Supervisor and Programmer roles. The Supervisor' and Programmer' roles are not assigned any privileges and can inherit all the privileges from the Supervisor and Programmer roles, respectively. Thus the Supervisor' and Programmer' roles possess the set of timing constraints and privileges originally associated with the Supervisor and Programmer roles, respectively. While the original information of the Supervisor and Programmer roles is retained, the Manager role can still

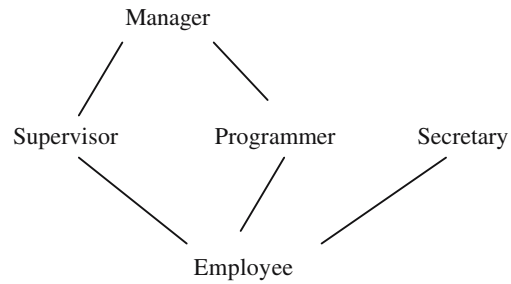

Fig. 7 An example of general role hierarchy

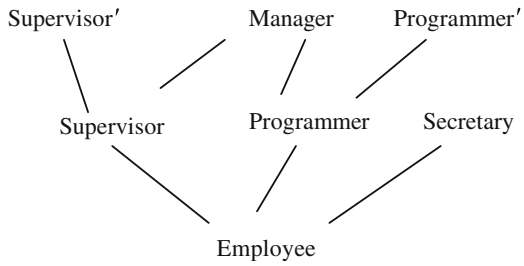

Fig. 8 Limited inheritance

inherit their privileges without any conflicts due to the newly specified timing constraints.

4.2.3 Authorization Decision Statement in the CAS Credential

Once the roles and their privileges and constraints are specified in the CAS database as described above, a SAML authorization decision statement is included in the CAS credential. An example of the SAML authorization decision statement is shown in Fig. 9, and its components, denoted by (1)–(4), are explained as follows:

(1) specifies the time period during which the authorization decision is valid.
(2) specifies the URI of the resource on which the permissions are granted.
(3) specifies the identity of the user to whom the permissions are granted.
(4) specifies what permissions are granted.

The user identified by the Subject in (3) is authorized in the role "Alpha,RN=Manager" with the privilege to execute the UPDATE operation on the resource specified in (2). Also, "Alpha,RN=Manager" inherits, from its junior role "Alpha,RN=Supervisor,"

Fig. 9 SAML authorization decision statement issued by CAS

```
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
AssertionID="8b53a37e-3116-44e2-a499-67e2d0fe49f1"
IssueInstant="2005-12-02T19:58:23Z"
Issuer="O=Grid,OU=GlobusTest,OU=simpleCA-
motive.cs.wright.edu,CN=Globus Simple CA" MajorVersion="1"
MinorVersion="0">

<Conditions NotBefore="2005-12-02T19:58:23Z" NotOnOrAfter=
"2005-12-02T21:20:53Z"></Conditions>                          (1)

<AuthorizationDecisionStatement Decision="permit"
Resource="http://130.108.17.176:8080/ogsa/services/ogsadai/
SecureGridDataServiceFactory.Employee">                       (2)

<Subject> /O=Grid/OU=GlobusTest/OU=simpleCA-                  (3)
motive.cs.wright.edu/OU=cs.wright.edu/CN=Vineela Muppavarapu
</Subject>
……..
<Action Namespace="role">Alpha,RN=Manager</Action>
<Action Namespace="Alpha,RN=Manager">privilege:
update</Action>
<Action Namespace ="Alpha,RN=Manager">junior_role:
Alpha,RN=Supervisor</Action>                                  (4)

<Action Namespace="Alpha,RN=Supervisor" >privilege:
select</Action>
<Action Namespace="Alpha,RN=Supervisor">timing_constraint:
gmt#10.01.2005-07.30.2006#MON-FRI#19:00-5:00</Action>

</AuthorizationDecisionStatement>
……..
```

the privilege to execute the SELECT operation on the same resource with the specified timing constraint "gmt#10.01.2005–07.30.2006#MON–FRI#19:00–5:00." The timing constraint specifies the duration for which the user can access the resource in the authorized role. This authorization decision is valid for the time period specified in (1).

4.2.4 Enforcement of VO Policies

In our method, the decision to map a VO role to a local role lies in the hands of the resource provider. The assignment of privileges to the local role and specifying timing constraints on it will also be the responsibility of the resource provider.

For example, as shown in Fig. 10, a resource provider can decide to map the ER,RN=Physician role, where ER could be an Emergency Team that forms a VO across several hospitals, to a local role that allows the SELECT and UPDATE operations to be performed between 19:00 and 5:00 GMT, Monday through Friday during 10.01.2005–07.30.2006 (say, only for those patients affected by a natural disaster). This timing constraint could be enforced by a database trigger, which executes an action automatically on the occurrence of a predefined event. The privileges and constraints associated with the local role can be negotiated between the VO and the resource provider. Alternatively, if local privileges and constraints have been fixed already, they can be made known to the VO or advertised through the service data of the Grid Data Service Factory (GDSF). The GDSF service data provides information about the underlying data resource, such as the database schema and the activities permitted.

The VO can restrict the policies further by specifying a subset of the privileges associated with the local role and/or specifying tighter constraints. For example, applications invoked by users in a Junior Physician VO role may be allowed to perform SELECT and UPDATE operations only between 21:00 and 5:00 GMT, instead of between 19:00 and 5:00 GMT. This scheme allows the VO to change privileges and constraints without involving the resource provider. However, these changes have to be enforced at the VO level. For example, the user's query and the current time can be examined in order to check the conformance with those changes. In this way, the resource provider does not have to create new local roles in addition to existing ones because both the original and restricted VO roles can be mapped to the same local role.

Furthermore, the resource provider can enforce more restrictions in addition to those imposed by the VO policy; for example, restricting the access privilege of particular users based on their institutional affiliation. For this purpose, the resource provider can maintain a separate list of users and deny their access by checking the Grid identity present in the CAS assertion. This enables the resource provider to have the ultimate authority over its resources.

5 Implementation Details

CAS has a backend database for storing information about users, resources and associated privileges. The VO members are granted user credentials signed by a Certificate Authority (CA). CAS issues a certificate to

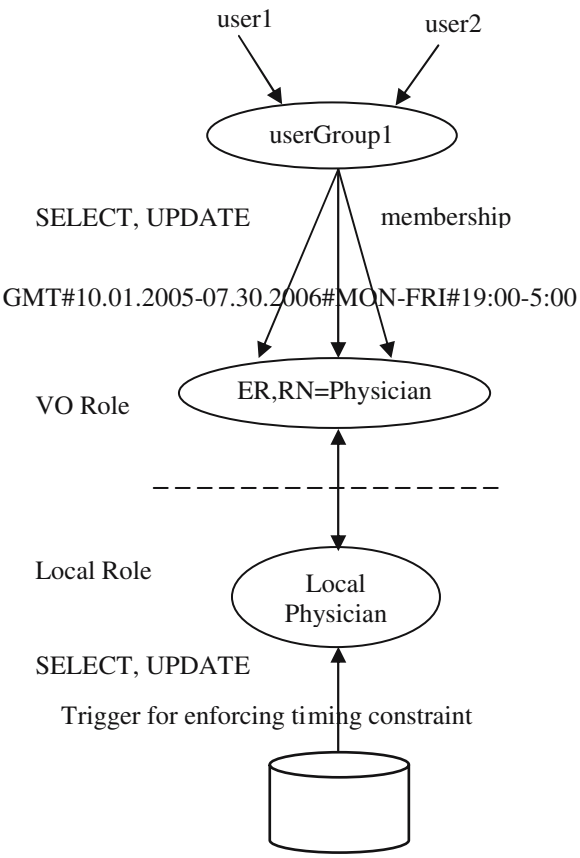

Fig. 10 Specifying VO roles using CAS

authorize users based on their requested role, their user credentials and the role membership information in the CAS database. The CAS database administrator can delegate the right to grant/revoke memberships on roles to other users, and those users can exercise that right only within the user groups to which they belong.

CAS provides a set of APIs for managing fine-grain access policies for resources in a VO [17]. The Service API of CAS provides an administrative interface for managing the user groups and associated privileges. This API supports the user's role assignments in our method. CAS also provides a Client API through which users can obtain a signed SAML assertion and present it to the resource provider for authorization. The OGSA-DAI client program uses the Java Generic Security Services API (GSSAPI) to delegate the CAS credential to a Grid Data Service (GDS).

We configured CAS to incorporate the proposed RBAC method as described before and modified the OGSA-DAI implementation to make use of the CAS credentials. The modifications are made at both client-side and server-side. The client is modified to delegate the CAS credential instead of the user proxy credential. The server is modified to recognize the CAS credential delegated by the client and to obtain the VO role from it using the GSSAPI libraries. The modified server also verifies the privileges and constraints associated with the VO role against the local policy, and performs the mapping based on that role via the role-map file. The role-map file has been extended to include the mapping from a VO role to a database username and a password. Also included in it are the local policy details as shown in Fig. 11. The role-map file can also include a list of users for whom access would be denied based on their Grid identity.

In order to use a CAS credential, the user initiates a CAS proxy by making a request to the CAS server based on the user's proxy credential. The CAS server authenticates the user and issues policy assertions based on the user's capabilities present in the CAS database.

As shown in Fig. 12, once the user has obtained the CAS credential with the requested assertions, the user can contact the desired GDSF to create a GDS. The GDS gets the CAS credential delegated by the user, and verifies the capabilities against its local policy present in the role-map file. The GDS also checks if any specified timing constraint is violated.

Figure 13 depicts a typical user session using the command-line tools provided by the Globus Toolkit, CAS and OGSA-DAI, which shows the initiations of the user proxy and the CAS proxy. We have modified the OGSA-DAI client to accept the CAS credential

Fig. 11 Modified role-map file

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--  (c) International Business Machines Corporation, 2002 - 2004.  -->
<!--  (c) University of Edinburgh, 2002 - 2004.   -->
<!-- See OGSA-DAI-Licence.txt for licencing information.   -->
<DatabaseRoles>
   <Database name="jdbc:mysql://130.108.17.176/ogsadai">
     <User dn="No Certificate Provided" userid="ogsadai" password="ogsadai" />
     <User dn="/O=Grid/OU=GlobusTest/OU=simpleCA-
     motive.cs.wright.edu/OU=cs.wright.edu/CN=Vineela Muppavarapu"
     userid="ogsadai" password="ogsadai" />

     <User dn=ER,RN=Physician" userid="username" password="password" />

     <Role Name="ER,RN=Physician">
       <Action Namespace="role">privilege:select </Action>
       <Action Namespace="role">privilege:update</Action>
       <Action Namespace="role"> timing_constraint:GMT#10.01.2005-
       07.30.2006#MON-FRI#19:00-5:00</Action>
     </Role>
   </Database>
</DatabaseRoles>
```

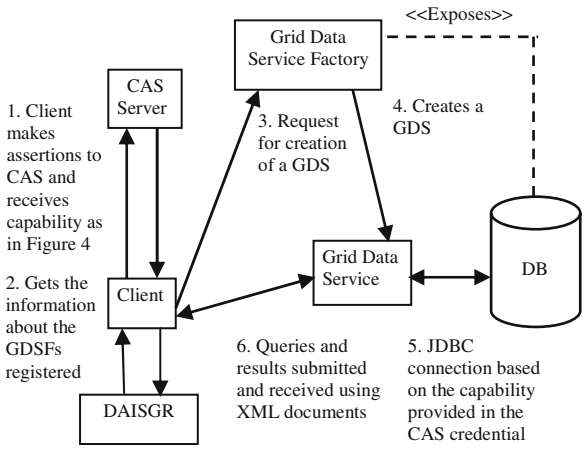

Fig. 12 Accessing a data resource through OGSA-DAI using a CAS credential

and the desired VO role specified as shown in Fig. 13. Based on the VO role of the user, a JDBC connection is established between the GDS and the database exposed by the GDSF. If no role is provided in the CAS credential, then the user's identity is used for mapping. The client can submit queries to the GDS and obtain the results in XML documents.

In our implementation, the GDS checks the local policy in the role-map file against the policy assertion in the CAS credential only before connecting the client to the database. After a CAS credential is issued, if a set of privileges is deleted from a VO role on the CAS server and/or the timing constraints on the role are changed, then the credentials have to be expired before the new policy takes effect on the resources. We have not implemented mechanisms to revoke current credentials containing old policy assertions. However, if the same set of privileges deleted from the VO role is also deleted from each of the corresponding local roles, or the same changes to the timing constraints are made, then the access to the resources can be restricted immediately based on the new policy. The resource providers can use the local database management systems (DBMSs) to update the privileges on the local database roles and modify the triggers to accommodate the new timing constraints. The changes in the local policy information also have to be made in the role-map files. Since the privileges and timing constraints on the local database roles are enforced by the local DBMS itself, they will come into effect immediately. If the client submits a query, but the query fails due to the new policy, then the client can be notified via an error message. The client can request new credentials and then restart the application.

If a VO role is updated independently of the corresponding local roles after the credentials are issued, one way to restrict the access immediately is to have the CAS server notify the GDSFs of the updates. This information can be passed on to the GDSs and cached by the GDSFs for up to the maximum lifetime of the credential. Any credential issued before the notification and containing an authorized VO role, which has been updated on the CAS server, can then be rejected. If a connection to

Fig. 13 User session accessing a GDS using CAS.

```
#Initiate a User Proxy
% grid-proxy-init
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-
motive.cs.wright.edu/OU=cs.wright.edu/CN=Vineela Muppavarapu
Enter GRID pass phrase for this identity:
Creating proxy... Done
Your proxy is valid until: Fri Dec 2 21:20:53 2005

#Initiate a CAS Proxy
%cas-proxy-init -c
http://localhost:8080/ogsa/services/base/cas/CASService -t tag

#Contacting a specific GDSF using CAS capabilities
%java uk.org.ogsadai.client.Client -mls -role ER,RN=Physician -t tag -factory
http://130.108.17.176:8080/ogsa/services/ogsadai/SecureGridDataService
Factory examples/GDSPerform/JDBC/query/select1Row.xml
```

the resource has been already established based on that role, then it should be discontinued.

If the local policy is changed to deny the access of a particular Grid user after that user has already been connected to the resource, then this new policy is not enforced because the role-map file is not rechecked. One possible solution to enforce the new local policy immediately is to notify the GDS every time the local policy information in the role-map file is updated, so that the user identity and policy assertion in the CAS credential can be checked against the local policy information.

With our method, a user who wants to perform the tasks associated with multiple roles does not need to generate multiple CAS proxies. The user can just delegate a single CAS credential containing all those roles. For example, a user may want to read from one database in one role and write to another database in another role. In this case, a single CAS credential containing both roles can be delegated, and then the user can be authorized by each resource provider with respect to the corresponding role.

6 Performance Analysis

The existing implementation of the OGSA-DAI client has been modified to delegate a CAS credential, and the server has been modified to obtain the user's capabilities present in the CAS credential. The overheads incurred with our implementation are compared with those of the existing implementation of OGSA-DAI, which does not use the CAS credential. OGSA-DAI Release 4.0 was deployed on a Jakarta Tomcat 5.0.27/Globus Toolkit 3.2.1 (GT3) stack running on a Linux machine with a 2.6 GHz Intel Pentium IV processor and 1 GB of RAM. The *littleblackbook* MySQL database table distributed with OGSA-DAI was used as a test database, and it contains 10,000 tuples. The *perform* document consisting of a request for a single tuple was used for the purpose of analysis.

6.1 Profiling Details

A Java method *System.currentTimeMillis*() is used to get the current system time in milliseconds. Also, for the server-side analysis, the Apache Log4j logger, which logs time to a log file in milliseconds, is used. For more accuracy, the tomcat container was shutdown and restarted before each client request in order to minimize the caching effects within GT3 and OGSA-DAI [23]. The main changes from the original configuration are the way the mapping is done at the server-side and how the credential is delegated at the client-side. So, only the security aspects of the client and the server are profiled and analyzed. The following types of Grid Data Services (GDSs) are used in the analysis as in [23]:

1) *Signature*: GDS enforcing GSI Secure Conversation with Signature. This enforces message integrity being established between the client and the server.
2) *Encryption*: GDS enforcing GSI Secure Conversation with Encryption. This enforces message privacy being established.
3) *None*: GDS which does not enforce any security. The GDS does not provide a secure conversation.

6.2 Client-Side Security

A call is made to each of the above GDSs with and without using a CAS proxy credential. In case of using a CAS proxy credential, an additional overhead for its creation is incurred. In the performance analysis, we do not show this overhead because it is incurred only once before the client contacts the GDSs. Thereafter, the client can submit any number of queries before the CAS proxy credential expires. The lifetime of the CAS proxy credential is equal to the time remaining for the expiration of the user proxy credential, which can last up to 12 hours. The time taken for the creation of the CAS proxy credential depends on several factors such as network bandwidth and workload of the CAS server. In our system, the average time taken for the creation of a CAS proxy credential is around 600 ms.

The *findServiceData* method of a GDSF returns the information about its corresponding data resource. Three consecutive calls to *findServiceData* are required: the first call returns the database schema, the second returns the activities permitted, and the third returns the product type (for example, the type of DBMS). The *perform* method of a GDS takes the *perform* document, which contains the query, and returns the results to the client.

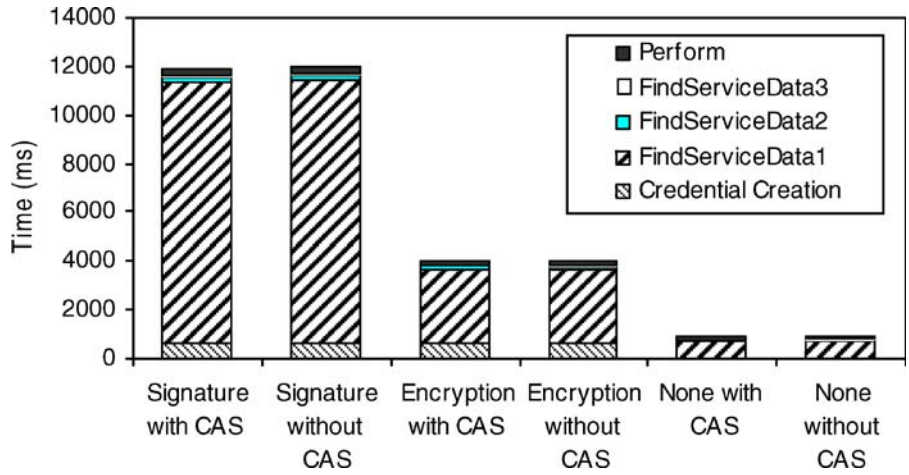

Fig. 14 Client-side security

GSI Secure Conversation requires a security context to be established between the client and the server. The overheads incurred in setting up this security context are analyzed based on the following:

1. Calls made for creating a credential object from the proxy credential.
2. Calls to the *findServiceData* and *perform* methods.

The corresponding times are shown in Fig. 14, and as observed, the time for creating the credential object is almost the same regardless of the security enforced by the GDS. In case of *None*, there is no such overhead as the credentials are not used. The first call to the *findServiceData* takes longer than the subsequent calls because it includes the initialization of the GDS regardless of the security type used. Figure 15 clearly shows the times taken for the subsequent calls to the *findServiceData* and the *perform* methods. The times recorded in the case of using a CAS proxy credential and those without using a CAS proxy credential are almost the same. The reason is because all the security functions on the client-side remain unchanged except for the use of a CAS proxy credential instead of a user proxy credential.

6.3 Server-Side Security

The analysis made on the server-side is based on the following:

1. The client credentials accessed using the GT3 infrastructure.
2. Extracting the VO role or Grid identity from the credential. If the VO role is extracted, its capabilities are compared against the local policy.

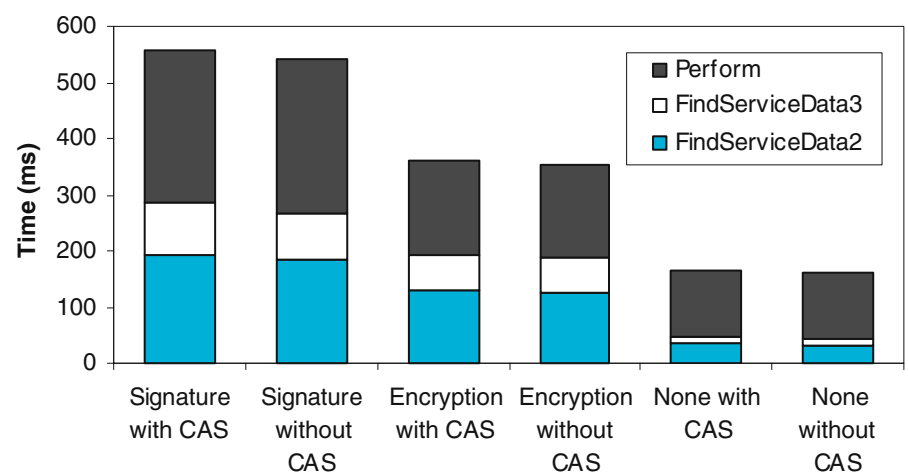

Fig. 15 Obtaining service data and query execution

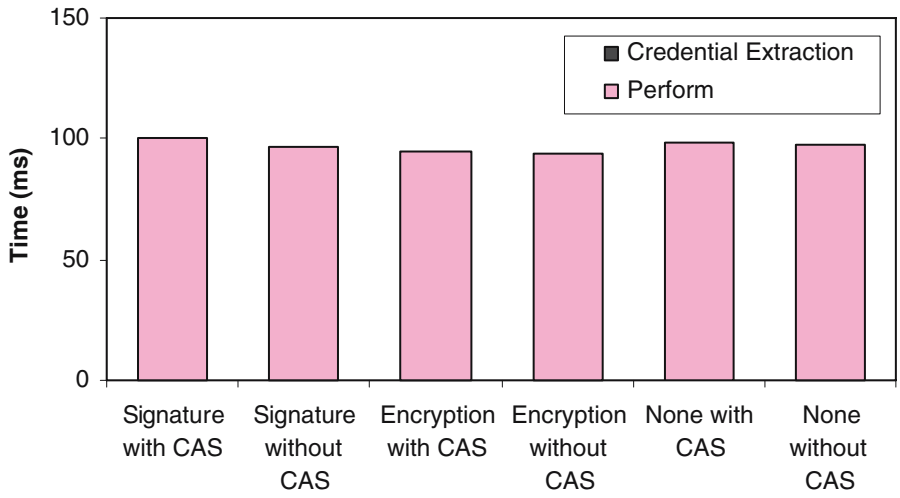

Fig. 16 Server-side security

3. Mapping a user to a database username and a password, and creating a JDBC connection.
4. The *perform* operation.

As shown in Fig. 16, the time for the credential extraction, which includes policy comparison, is very small compared to the time for executing the *perform* operation. The time for executing the *perform* operation remains constant for all the GDSs. The *perform* operation is done only after the credential extraction process is completed; and as a result, its execution time is not affected by the type of credential used. The credential extraction times are shown more clearly in Fig. 17, and we can see that the credential extraction takes more time when a CAS proxy credential is used for contacting a GDS that enforces the secure conversation. An overhead is incurred because of the time taken for obtaining the user identity and the policy assertion from the CAS credential and then comparing it against the local policy in the role-map file. However, this overhead is in the order of a few milliseconds and is insignificant compared to the overall time taken for performing the client's query. When a CAS proxy credential is not used, the user proxy credential is used instead, and then the credential extraction involves obtaining only the user identity. In case of contacting a nonsecure GDS, since credentials are not used, there are no overheads incurred for credential extraction.

Figure 18 shows that there is a constant overhead for mapping a user to a database username and a password and then subsequently setting up the database connection. The processes of mapping and connection are done after the credential extraction

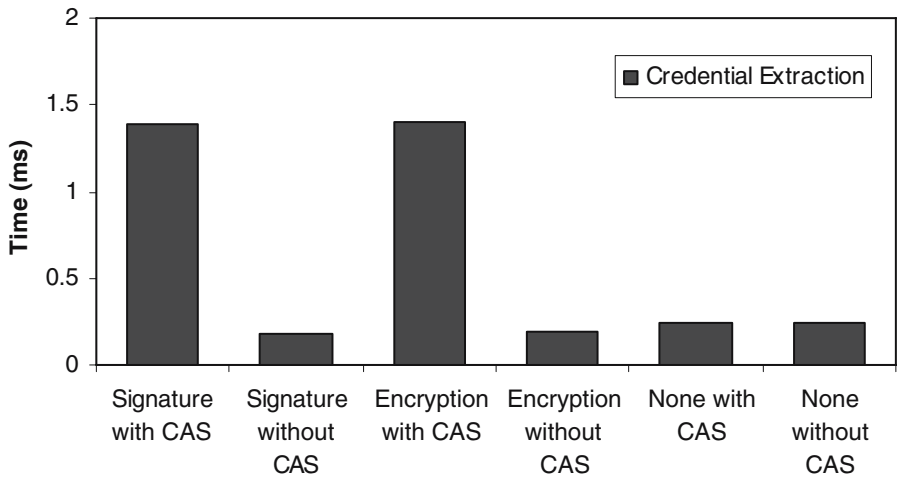

Fig. 17 Security overheads on the server-side

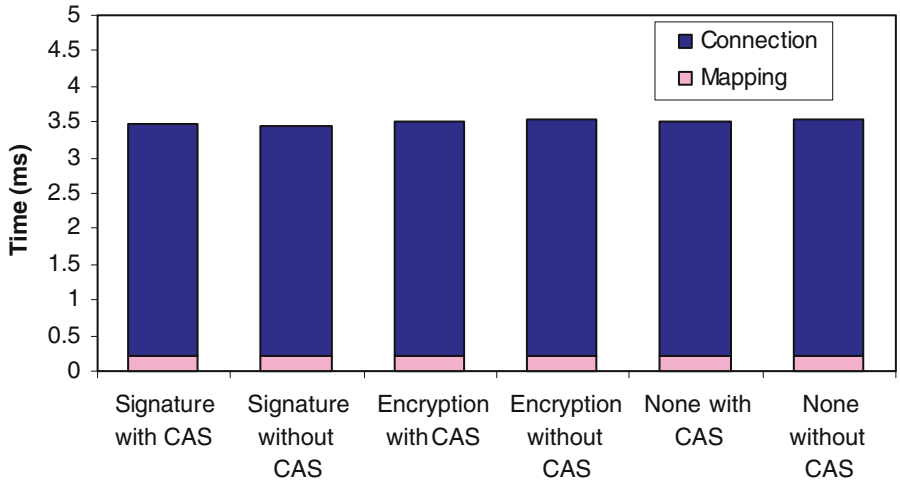

Fig. 18 Mapping and database connection

process is completed; and as a result, their execution times are not affected by the type of credential used. If there are a large number of entries in the role-map file, the mapping would still not take much time because a hash table is used to store those entries.

In summary, on the client-side, our method incurs small overheads in the security setup as additional steps are involved for requesting and using the CAS credential. However, as seen from the performance results, the time taken for the individual OGSA-DAI method calls are the same whether a CAS credential is used for authorization or not. This is because all the security functions remain unchanged, except for the use of a CAS proxy credential instead of a user proxy credential. On the server-side, the additional overheads incurred in our credential extraction process are very small compared to the time taken for executing the client's queries. These overheads in setting up the security context are insignificant when we consider the benefits of our method, such as scalability in managing VO policies and reduced administration overheads for resource providers.

7 Conclusion

In this paper, we enhanced the role-based access control (RBAC) mechanism of OGSA-DAI by using the Community Authorization Service (CAS) so that users are granted memberships on virtual organization (VO) roles for Grid database services. The resource providers need to maintain only the mapping information from VO roles to local database roles and the local policy information; thus, the number of entries to be managed in the role-map file is reduced dramatically. The specification of policies at the VO level eliminates unnecessary authentication, mapping and connections by denying invalid requests at the VO level itself. When users join/leave a VO, the resource providers do not need to add/remove their information individually in the role-map files because the CAS server can just grant/revoke their memberships on VO roles. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately in the role-map files. This enables the resource providers to have the ultimate authority over their resources.

Our performance analysis shows that the proposed RBAC method using CAS takes very little extra time to set up the security context between the client and the server. Our method provides a scalable means of access control in terms of the manageability of a large number of users and VOs. It also has an advantage of reducing the administration overheads of resource providers.

Acknowledgement This research was supported in part by AFRL.

References

1. 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)
2. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: Grid services for distributed system integration. IEEE Computer **35**(6), 37–46 (2002)

3. Camarinha-Matos, L.M., Afsarmanesh, H.: A roadmap for strategic research on virtual organizations. In: Proceedings of the 4th IFIP Working Conference on Virtual Enterprises, Lugano, Switzerland, pp. 33–46 (2003)
4. Arenas, A.E., Djordjevic, I., Dimitrakos, T., Titkov, L., et al.: Toward web services profiles for trust and security in virtual organizations. In: Proceedings of the 6th IFIP Working Conference on Virtual Enterprises, Valencia, Spain, pp. 26–28 (2005)
5. Wasson, G., Humphrey, M.: Policy and enforcement in virtual organizations. In: Proceedings of the 4th International Workshop on Grid Computing, Phoenix, Arizona, pp. 125–132 (2003)
6. Wasson, G., Humphrey, M.: Towards explicit policy management for virtual organizations. In: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, pp. 173–182 (2003)
7. Malaika, S., Eisenberg, A., Melton, J.: Standards for databases on the Grid. ACM SIGMOD Record **32**(3), 92–100 (2003)
8. Ferraiolo, D., Kuhn, R.: Role-based access control. In: Proceedings of the 15th National Computer Security Conference, Baltimore, MD (1992)
9. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer **29**(2), 38–47 (1996)
10. 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)
11. Foster, I., Kesselman, C.: The Globus Toolkit. In: Foster, I., Kesselman, C. (eds.) The Grid: Blueprint for a New Computing Infrastructure, pp. 259–278. Morgan Kaufmann, San Francisco, CA (1999)
12. Butler, R., Welch, V., Engert, D., Foster, I., Tuecke, S., Volmer, J., Kesselman, C.: A National-scale authentication infrastructure. IEEE Computer **33**(12), 60–66 (2000)
13. Anjomshoaa, A., Antonioletti, M., Atkinson, M., Baxter, R., Borley, A., et al.: The design and implementation of Grid database services in OGSA-DAI. In: Proceedings of UK e-Science All Hands Meeting, Nottingham, UK (2003)
14. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Vanderbilt, P.: Grid Service Specification, Draft 4. Open Grid Service Infrastructure Working Group, Global Grid Forum (2002)
15. Joshi, J.B.D., Bhatti, R., Bertino, E., Ghafoor, A.: Access-control language for multidomain environments. IEEE Internet Comput. **8**(6), 40–50 (2004)
16. Mayfield, T., Roskos, J.E., Welke, S.R., Boone, J.M.: Integrity in automated information systems. Technical Report, National Computer Security Center (1991)
17. 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 (2002)
18. 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)
19. Ferraiolo, D.F., Barkley, J.F., 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)
20. Zhang, G., Parasher, M.: Dynamic context-aware access control for Grid applications. In: Proceedings of the 4th International Workshop on Grid Computing, pp. 101–108 (2003)
21. Cannon, S., Chan, S., Olson, D., Tull, C., Welch, V., Pearlman, L.: Using CAS to manage role-based VO sub-groups. In: Proceedings of International Conference for Computing in High Energy and Nuclear Physics (2003)
22. 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)
23. Jackson, M., Antonioletti, M., Hong, N. C., Hume, A., Krause, A., Sugden, T., Westhead, M.: Performance analysis of the OGSA-DAI software. In: Proceedings of UK e-Science All Hands Meeting, Nottingham, UK (2004)
24. Yee, K.: Secure interaction design and the principle of least authority. In: Proceedings of Workshop on Human–Computer Interaction and Security Systems, Fort Lauderdale, FL (2003)