Nepheli: Scalable Decentralized Authorization in Federated File Services

G. Margaritis A. Hatzieleftheriou N. Mpountouropoulos S. V. Anastasiadis
Department of Computer Science
University of Ioannina
Ioannina 45110, GREECE
{gmargari,ahatziel,nmpountou,stergios}@cs.uoi.gr

Technical Report DCS 2010-03
April 19, 2010

ABSTRACT

We consider the problem of remote access control in federated file services across independent administrative domains. Existing approaches either establish complex trust relationships between distinct domains or require frequent update operations among different authentication and authorization services. In order to improve the manageability and scalability of current solutions, we propose to specify remote access rights on the basis of federation-wide user groups (*virtual groups*) instead of individual users. The members of a virtual group are users that originate from different domains and undertake a common role in the federation. The realization of a virtual group has a two-layer structure, where the upper layer consists of domains, and the lower layer consists of users contributed by the participating domains.

We certify the domain membership of a virtual group at the authentication service of the domain that creates the group. User access requests carry an extended certificate that is issued by the local authentication service and encloses a list of the virtual groups whose members include the user. In order to authorize a remote access, the resource provider simply applies its local policy that pertains to the virtual groups of the requesting user. We achieve low maintenance cost because the remote update traffic only distributes the domain membership of virtual groups to the user authentication and the resource authorization services across the federation. We implement a prototype of our architecture and experimentally demonstrate reduced cost and improved scalability properties.

1. INTRODUCTION

We examine the frequent need of independent organizations to form federations and contribute storage resources for mutual data sharing among their users. Such infrastructures can form the basis for flexible collaboration environments that serve a wide range of applications in areas such as business, education, science and engineering [2–4, 8, 9, 13, 21]. Existing solutions for resource peering among independent organizations are not easily applicable to fine-grain data accesses because the access control overhead may exceed the cost of the remote data access. We assume that each organization is an independent administrative domain (or simply *domain*) with separate security policy. The domain hosts users, supports user groups, and operates resource providers to handle resource access requests. Each domain maintains a local authentication service to issue certificates that prove the identity of the local users and certify attributes such as the groups that have a particular user as member. Similarly, every resource provider relies on a local authorization service to manage the access control attributes that determine the permissions of the local access requests.

In the case of remote access control, the main problem that arises is the authentication of the remote users and the authorization of their requested accesses. One approach to solve this problem is to have the administrator of each domain manually add remote users to the local authentication and authorization services [4]. As a result of the administration overhead of this approach, it is only applicable to small-scale installations where user participation is infrequently modified. An alternative strategy would automatically have the user domains export their user group structure to the resource providers [9]. Practically, this involves periodical transfer of the membership list of each group from the authentication service of the user domain to the authorization service of the resource provider. Due to the potentially large number of users and groups involved, this solution may incur sensitivity to network transfer delays and disconnections.

Another way to allow in a domain remote data accesses is to use hierarchical trust connections and key exchanges between different domains [3, 14]. However, the establishment of trust relationships may require complex legal and political negotiations between the participating organizations. In an effort to reduce the administration cost involved, it is possible to create a global authorization service that has full knowledge about the participating members of each domain and the particular resources that they can access [16]. This solution introduces periodic update overhead to keep the global service consistent with the individual organizations as user enrollment changes over time, or resource access permissions are adjusted. Additionally, there are latencies needed to let each member from an organization communicate with the global service to receive the necessary authorization certificates and then forward them to the remote site for the actual data transfer.

One of the challenges in remote access control is the hypothetical need to export the lists of user group membership from one domain to another. This is necessary if a resource provider specifies access permissions in terms of remote user groups and requires to know the current membership of each group in order to authorize remote accesses from individual users. To overcome this limitation, we define the *virtual group* as a federation-wide collection of users that undertake common role in the federation and they are contributed by individual domains. In its simplest form, a virtual group consists of two single-level layers; the upper layer is a collection of collaborating domains and the lower layer is formed by the collection of users that participate from each domain. More generally, the upper layer is a group hierarchy of domains, while the lower layer is a union of user group hierarchies contributed respectively by the participating domains.

The distinctive feature of a virtual group is that its lower-layer structure is not explicitly replicated across the collaborating domains. If a resource provider grants permissions to a virtual group, the resource provider only needs to know the particular domains that belong to the virtual group during an authorization decision. Correspondingly, a user domain certifies that a local user is member of specific virtual groups and attaches this membership certification to every remote access request. Similarly, the domain that creates a virtual group, periodically certifies the domain membership of the virtual group to the collaborating domains. Since the number of domains that participate in a virtual group is orders of magnitude lower than the number of participating users, the respective certification cost is much lower in comparison to alternative approaches that involve establishment of delegation hierarchies, or full disclosure of the group memberships across the federation.

The main contributions of the paper include:

- Definition of a collection of security requirements for distributed access control in federated file services,
- Specification of a security model based on the concept of virtual groups to meet the above requirements,
- Implementation of the Nepheli system prototype within the Globus Toolkit standard middleware environment, and
- Experimental evaluation of the performance properties of the Nepheli prototype in comparison to existing systems.

In the rest of the paper, we describe the Nepheli architecture that we propose in Section 2, while in Section 3 we present its prototype implementation. In Section 4, we outline our experimentation environment, and in Section 5 we analyze the measurements that we did to evaluate the performance of the system. In Section 6, we summarize our conclusions and plans for future work.

2. ARCHITECTURE

In the present section, we first define some basic concepts, then we introduce our assumptions about the requirements of remote access control in federated file services, and then we describe our design decisions in the system architecture that we propose.

2.1 Basic Definitions

We assume that each administrative domain is addressable through a globally unique identifier, such as the hostname of an authentication service that certifies entities of the domain. The hostname

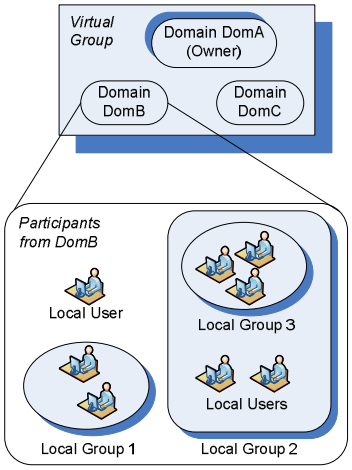


Figure 1: Each participating domain contributes to the virtual group an arbitrary hierarchy of local users and groups.

can be self-certifying in the sense that it contains a hash of public-key to facilitate mutual authentication with remote entities, as has been previously suggested by Mazieres et al. [12]. An organization creates a virtual group as a domain-specific unique name and an associated list of domains. The respective domain is called *owner* of the virtual group, and has its authentication service securely certify the attributes of the virtual group. Each participating domain contributes to the virtual group a subset of users with arbitrarily complex group structure, as shown in Figure 1. For instance, we may assume a joint project, called *fedstore*, among the domains `uoi.gr` and `unisalento.it`. If we assume that the domain `uoi.gr` is the project coordinator, we have this domain create the virtual group `admins.fedstore@uoi.gr` to keep track of the project administrators, `scientists.fedstore@uoi.gr` for the research staff, and `engineers.fedstore@uoi.gr` for the engineering personnel. Each virtual group is defined as a list of two domains `{uoi.gr, unisalento.it}`. Respectively, we leave responsibility of the domains to specify the local users and groups that belong to each of the above three virtual groups.

2.2 Assumptions

Policy distribution. The access-control policy specifies the user permissions across the offered services of the resource providers. Prior research has suggested the use of authorization certificates (*credentials*) to decentralize the mechanisms of policy specification and enforcement at different parts of a distributed system [10]. Then, it is possible to have user requests transfer security policies from the point of specification to the point of enforcement. In our model, we locate both functions of policy specification and enforcement locally at each individual resource provider. We use the concept of virtual group to add an extra abstraction layer in the specification of the users and correspondingly split the authentication service between the domain that hosts a user and the domain that owns the virtual group. Our objective is to minimize the amount of state information that is transferred between the collaborating domains during periodical updates or access requests.

Administration. We anticipate the number of enrolled users across a federated system to be much larger than the number of partici-

pating domains. Consequently, the queries to certify the domain membership of a virtual group should be infrequent in comparison to queries that certify the list of virtual groups that have a user as member. An access request carries to the resource provider the list of virtual groups of the user, while the resource provider periodically pulls or receives the domain membership of each referenced virtual group from the corresponding owning domain. Essentially, the authentication of a user occurs at the domain of the user, while the request authorization remains responsibility of the resource provider. The ability to transfer rights from one user to another is useful to easily establish collaborations with arbitrary remote users and make available to them local resources, such as data files [3]. In our model, we limit the administrative overhead required to transfer access rights to remote users but we don't completely eliminate it. The administrator of a domain has to approve the participation of the domain along with its local users to a virtual group created remotely before the local users can access remote resources granted to the virtual group. The alternative approach of directly granting permissions to a remote user adds flexibility to the system at the cost of increased update overhead.

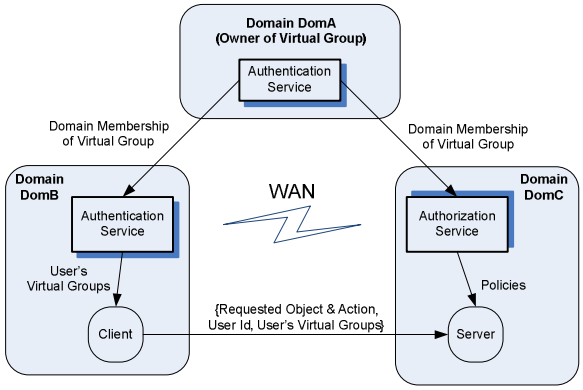


Figure 2: The Nepheli architecture keeps the identity authentication of the clients separate from the authorization of their requested actions at the server. The domain membership of the virtual group is the only authentication attribute that is not carried by the user's request, but instead requires certification directly from the domain that owns the virtual group.

Access granularity. The granularity of data accesses specifies the relative impact of the incurred authentication and authorization cost. As a result, system designers follow distinct paths to support file downloads in comparison to block-based accesses. Usually, downloads are more common over wide-area networks, while block-based accesses are limited to small distances within a building or a campus. Since fine-granularity requests that occur in block-based accesses are more frequent, a lightweight management is needed to keep their service practical. As the capacity of the backbone network improves, the above distinction is blurred with the block-based accesses increasingly required over long distances. It remains a challenge to enable their secure operation without compromising their responsiveness and low-cost handling [15].

2.3 Design Issues

The virtual group establishes a context for resource sharing between independent organizations. It connects groups from different organizations that contain users with common role and authorization requirements. At abstraction level, the concept of virtual group lies between a federation of domains and a user group of a single domain.

Authentication. Each domain uses a public-key infrastructure to manage certificates and eliminate the need for key distribution from a global authority [5]. Thus, a user communicates with the local authentication service to receive an *identity* certificate signed by the private key of the service. The certificate associates the identity of the user with her public key. The certificate is only valid for a specific time period (*lease*), which limits the exposure of the system to unexpected security compromises. Subsequently, the user uses the identity certificate at the authentication service to request an *attribute* certificate that can be used for a local or remote access. In addition to the user's identity and home domain, the attributes that accompany a remote request include a list of virtual groups that have the user as member. We assume that virtual groups have globally unique names that can be certified by their owning domain, while the names of the remote services are distributed through the domain name registry.

As an optimization to reduce the volume of authentication data transmitted over the network and the cost of authorization processing at the remote domain, the access request may only carry the list of virtual groups that contain both the local and remote domain. The owner of the virtual group is responsible to certify the domain membership to the resource provider. For improved responsiveness and tolerance against network outages, a resource provider may cache the memberships of the virtual groups that have been recently used across received requests. This is similar to what existing systems do for memberships of user groups [9]. The advantage of our security model is that it keeps the main authentication functions local across the domains that own virtual groups, or accommodate resource providers and resource users (Figure 2).

Authorization. The resource providers at each domain maintain access-control lists to specify the permissions granted to local users and groups of the domain or virtual groups of the federation. Access-control lists are commonly used in file systems because they provide flexibility in access permission specifications and user accountability in comparison to alternative choices such as capabilities [13]. Therefore, when a remote request arrives, the resource provider compares the received attributes against the access-control list of the requested file. The request is served if at least one of the user's virtual groups is assigned the needed access permission. The number of virtual groups in a federation depends on the number of distinct roles required by the undertaken joint effort. We use the term role to describe a distinct collection of permissions granted by a resource provider to a collection of local and remote users to perform a particular task [18].

Revocation. In our model, the revocation of user rights is implicitly enforced through the lease expiration of the authentication certificates required for an access. More specifically, lease expiration is applicable to the attribute certificate of the user that contains her virtual groups and also the membership certificate of each virtual group issued by the corresponding owning domain. This is an inexpensive way to propagate updates about the domains that participate in each virtual group and the list of users contributed by each domain. Additionally, our model can also support the explicit distribution of revocation lists from authentication services to

resource providers in order to enable faster revocation of the participation of a user or domain in a virtual group. Such a solution provides immediate enforcement of policy to the resource providers, but increases the system cost and complexity.

Delegation. The right of a user to access a remote resource is granted through the addition of the user to the appropriate virtual group by the local security administrator of the user. We leave to the local security policy the possibility to support the autonomous delegation of membership rights from one user to another within a domain [5, 9]. However, delegation to a remote user requires first the addition of the remote domain to the virtual group. In the common case, domains form federations to serve specific collaboration needs that can be met by specific participants from each domain and respective roles in the collaboration. We expect that reasonable assignment of users to virtual groups by the administrators at each domain shall handle adequately well the access requirements that will emerge during a collaboration.

Accountability. Users are accountable for the requests that they make and the resources that they actually use across the different domain. This is enforced by having the identity of the user embedded in the certificates carried over each request and signed in a non-repudiable way by the authentication service of the user's domain. Thus, we can audit the consumption of resources at each remote domain by a user, and trace back the source of system misbehavior to the user that causes it.

2.4 Summary

We can outline the advantages of our approach as follows:

- *Decentralization.* There is no need for a central authority to know all users or accessible resources. The ownership of virtual groups is distributed across the domains similar to the way that users are.
- *Freshness.* Domain population changes are reflected into user requests according to the frequency at which users refresh their certification from the local authentication service.
- *Access Cost.* Requests only need to carry certified lists of virtual groups that simultaneously contain both the domains of the user and the resource provider.
- *Update Cost.* We reduce the update overhead at the resource providers because we locally cache the domain membership of recently used virtual groups. Thus, we also handle network disconnections and avoid complex policy distributions.

3. PROTOTYPE IMPLEMENTATION

We built a prototype of the Nepheli architecture based on the open source implementation of the Community Authorization Service (CAS) [16]. We use the GridFTP facility as a file service example to demonstrate the scalability of our approach, although we plan to include in our future work experimentation with finer-grain file accesses. In the rest of the present section, we describe the modifications that we applied to CAS and GridFTP in order to develop a prototype of the Nepheli architecture.

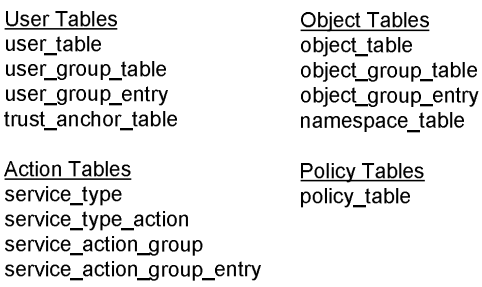


Figure 3: The relational database schema of the original CAS server consists of sets of tables that keep track of the users and their groups, the objects and their groups, the actions and their groups and the policies that associate the above to specify the permitted actions of the users over the objects.

3.1 The original CAS and GridFTP

The original CAS server includes a front-end SOAP-accessible web service written in the Web Service Definition Language, a back-end commodity relational database accessed through embedded SQL statements, and Java code that links these two components [6, 16]. The relational database centrally manages a collection of tables that keep track of users, resources, service actions and policies, as shown in Figure 3. The database schema specifies the certification authority of the users and organizes them into groups. It defines the supported types of service and structures the allowed actions into groups. Each accessible resource consists of objects also organized into object groups. Policy statements in the form of relational tuples associate the users with the objects and the corresponding permitted actions.

GridFTP refers to both a data transport protocol and a collection of client/server tools that allow fast transfer of files over wide-area networks [7]. The protocol has been designed to provide efficiency and reliability for bulky data transfers in a way that is compatible with standard security mechanisms. It improves performance and flexibility through parallel data transfers over multiple streams, and partial file transfers over wide-area links. The original CAS-enabled GridFTP client uses a short-term identity certificate to request from the CAS server a certificate extended with assertions of the resource access rights that apply to the user. Subsequently, the client forwards the extended certificate to the remote GridFTP server. The server uses the certificate to identify the user and applies the embedded access permissions to decide if the requested service action is allowed.

The CAS has been designed to scalably manage the access control policies across a federation by only establishing trust relationships between each domain and the CAS server instead of among all the domains between each other. The participating domains enroll into the central database all the contributed users and resources before they also register the access control policies. The central server can become bottleneck from the update and query traffic that it handles unless complex replication techniques are applied [16]. In addition, the user issues certificate requests that can be frequent or heavyweight depending on the number of policies that they carry. To overcome the above potential problems, we propose the Nepheli architecture that connects the federated domains through virtual groups, keeps each domain responsible to certify which virtual groups have local users as members, and has each resource provider specify access policies in terms of virtual groups.

```
/* Group directly contains user or user group */
SELECT DISTINCT user_group_name
  FROM user_group_entry
  WHERE (user_nickname = "userNick")
UNION
SELECT DISTINCT og.object_group_name
  FROM object_group_entry og, user_group_entry ug
  WHERE (og.object_specification =
                ug.user_group_name
          AND og.object_spec_desc = 'userGroup'
          AND ug.user_nickname = "userNick")
      OR (og.object_spec_desc = 'user'
          AND og.object_specification = "userNick");
```

(a)

```
/* Group indirectly contains user or user group */
SELECT DISTINCT object_group_name
 FROM object_group_entry
 WHERE object_spec_desc = 'object'
        AND object_specification IN ("group_list");
```

(b)

Figure 4: First, we invoke statement (a) to select groups that have as direct member either the user `userNick` or one of the user's groups. Then, we repeatedly invoke statement (b) to retrieve object groups that indirectly contain the user or the user's groups. The `group_list` refers to the list of groups returned initially by (a), or the previous invocation of (b).

3.2 The Implementation of Nepheli

We split the original CAS schema into two collections of tables that we respectively call the User Authentication Service (UAS) and the Action Authorization Service (AAS). Each domain has a local UAS to identify the local users and define the user membership of the local groups. Also, the UAS schema specifies the users and groups that belong to local and remote virtual groups. Correspondingly, each resource provider operates its own AAS to manage the exported resources, service actions and policies. We keep most of the related tables in AAS unmodified from the original CAS design, but we expand the policy specification to include action assignment to virtual groups. Thus, Nepheli relocates the definition of user groups from the central CAS server to the UAS of each domain and the specification of access policies from the central CAS server to the AAS of each resource provider.

The user initially communicates with a trusted certification authority to receive an X.509-based identity certificate in a way similar to the original CAS system. Then, the user applies the certificate to request from the local UAS an extended certificate that encloses a list of the virtual groups that have the user as member. The certificate extension is based on the Security Assertion Markup Language (SAML) [17]. The SAML standard allows an authority to issue assertions of security information for a subject. Each assertion contains statements which alternatively certify that the subject has been (i) authenticated at a particular time, (ii) associated with particular attributes or (iii) issued an authorization decision about the permission to access a particular resource.

The original Java implementation of CAS uses SQL statements to retrieve from the database first the user groups that have the user as member, then the policies that apply to them, and finally the groups of permitted actions. In order to prepare the requested SAML assertion, the CAS code expands the above data into a sequence of authorization decision statements that specify the individual actions permitted to each object for the requesting user. For the development of the Nepheli prototype, we implemented the UAS subsystem with Java code that retrieves the user groups based on the user identity. As shown in Figure 4, we use the SQL statement (a) once to select the groups that directly contain the user, and the SQL statement (b) multiple times to accumulate the groups that contain the user indirectly. We treat the virtual groups as a special type of user groups that appear at the upper levels of the hierarchy. We return them back to the user through the SAML assertion within an extended certificate.

```
/* Permitted actions over individual objects */
SELECT DISTINCT o.namespace_nickname, o.object_name,
        s.service_type_name, s.action_name
FROM policy_table p,
     object_table o,
     service_type_action s,
     service_action_group_entry se,
     object_group_entry oe
WHERE /* actions or action groups */
    (((p.action_spec_desc = 'serviceAction'
        AND p.action_specification =
            (CAST (s.service_action_id AS TEXT)))
      OR (p.action_spec_desc='serviceActionGroup'
        AND p.action_specification =
            se.service_action_group_name
        AND s.service_action_id =
            se.service_action_id ))
    /* objects or object groups */
    AND ((p.object_spec_desc = 'object'
        AND p.object_specification =
            (CAST (o.object_id AS TEXT)))
      OR (p.object_spec_desc = 'objectGroup'
        AND p.object_specification =
            oe.object_group_name
        AND oe.object_specification =
            (CAST (o.object_id AS TEXT))))
    /* user groups */
    AND p.user_group_name IN ("group_list"));
```

Figure 5: We retrieve all the allowed actions over objects that apply to the groups in `group_list`. This is an SQL statement invoked by the GridFTP server after the receipt of an extended certificate that contains the virtual groups of the user. The returned policies contain actions and objects that are specified either individually or organized into action and object groups, respectively.

The Nepheli user calls the GridFTP client to forward a file transfer request along with the received extended certificate to the GridFTP server. The server parses the SAML extension to extract the user's virtual groups. The server subsequently retrieves from the local AAS the access rights granted to the virtual groups. This is an operation that occurs online during the file request processing. Even though the above functionality was partly available in the original CAS implementation across approximately 9K lines (without the invoked libraries) of Java code, we needed it ported into the C implementation of the GridFTP server.

We considered several alternative approaches to (i) manually translate the above Java code into C, (ii) use the Java Native Interface to directly invoke the Java implementation from C, or (iii) expand the web service interface of CAS and have it directly used by the GridFTP server, as well. We disregarded the code translation due to the large amount of effort replication that it involved, and also decided to avoid the direct use of Java code by the GridFTP server due to skepticism about the performance overhead that it would introduce. Eventually, we decided to implement the required database access of the AAC through an SQL statement that would both select the relevant tuples and join them into the required list of fields (Figure 5). The use of a single statement to select tuples from five dif-

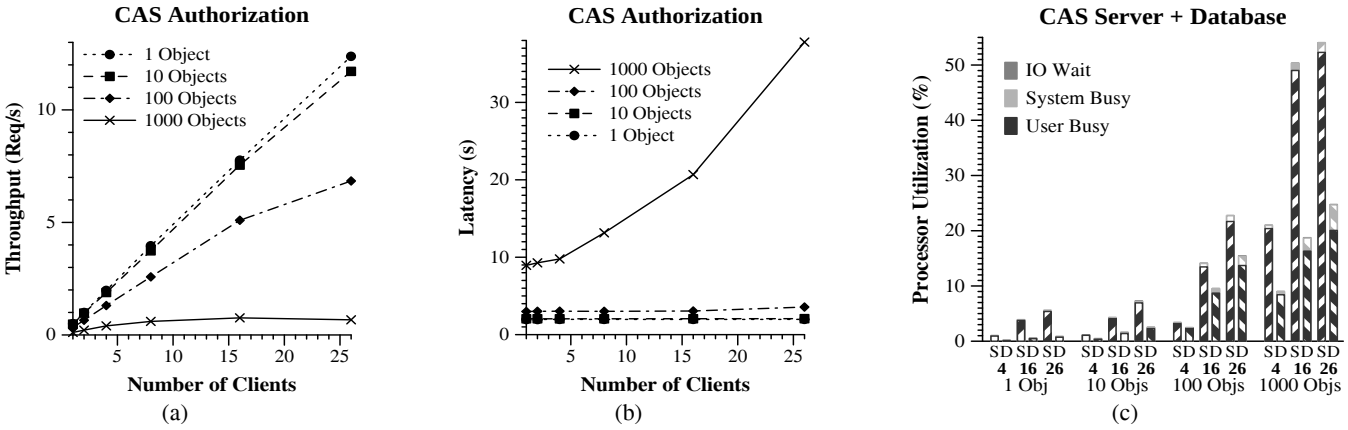


Figure 6: We examine the performance of a CAS server that issues extended authorization certificates for number of concurrent clients that changes between 1 and 26. We specify 5 action policies for each object. Then, for certificates with large number of objects (e.g. 1000), we observe that the server throughput (a) drops substantially and the request latency (b) increases to several tens of seconds, respectively. The heavy load is reflected into higher processor utilization (c) of the nodes that run the web service (S) and the database (D) of the CAS, as it is shown for numbers of clients 4, 16 and 26.

User Enrollment into CAS

Clients	*Latency (s)*	*Throughput (Req/s)*
1	1.6906	0.588235
2	1.6922	1.162791
4	1.6802	2.298851
8	1.6876	4.545455
16	1.6984	8.695652
26	1.7275	13.265306

Table 1: We measure the server throughput and request latency to enroll new users into the CAS web service. For this experiment, we ran multiple concurrent container processes on the CAS node. Even though latency remains almost flat and throughput scales linearly for the loads that we examined, overall, registering new users to a third-party through web service turns out to be a time-consuming procedure.

ferent tables and subsequently join them reduced substantially the required coding effort. We temporarily store the returned policies into a linked list of objects and allowed actions. Then, the server matches the objects and actions of the selected policies against the requested file and operation. In case of successful match, the server proceeds to the execution of the requested operation.

In a real deployment, we anticipate that the client and server domains are connected over wide-area links. Therefore, the certificate request to the UAS is served by the local domain of the user, while the authorization request to the AAS is served at the local domain of the file service. In the common case, the only needed interaction over the wide-area network is the communication of the file transfer request to the remote server along with the file transfer itself. The communication of the resource provider with a domain to certify the membership of a virtual group is similar to the certificate request of the user to the UAS with the only difference that the membership of virtual groups is expected to change less frequently. Thus, in our experimental measurements we focus on the cost of having the certificate request of a user served by the local UAS, and the cost of a file transfer request served by the GridFTP server in collaboration with the local AAS.

4. EXPERIMENTAL ENVIRONMENT

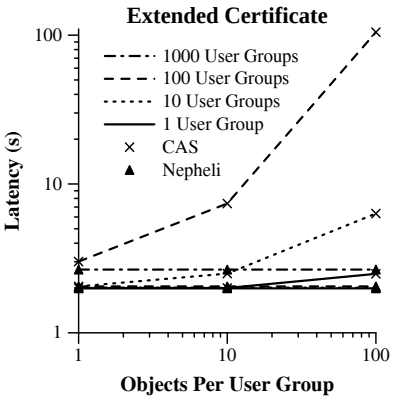
For our measurements, we used an isolated cluster of server nodes running the Debian distribution of Linux kernel version 2.6.18. Each node is equipped with one quad-core x86 2.33GHz processor, 2GB or 3GB RAM, one active gigabit ethernet port and two 7200RPM SATA disks of 250GB or 500GB each. The nodes are connected with a 48-port gigabit ethernet switch of packet latency 5.4μs and 96Gbps switching capacity. In our experiments, we used the open-source Globus Toolkit version 4.0.7, Postgresql object-relational DBMS version 8.4.0, Apache Ant version 1.7.1, and Java development kit version 1.6.0_14. We used one node to exclusively run the database, and another to execute alternatively the CAS, the Nepheli or the Gridftp service. We set the maximum number of threads in the web service of the CAS equal to 50, and the number of connections between the web service and the database equal to 100. We employed up to 26 separate nodes to generate client requests against the above services. In all cases, we show average numbers for the handling of 50 requests.

5. PERFORMANCE EVALUATION

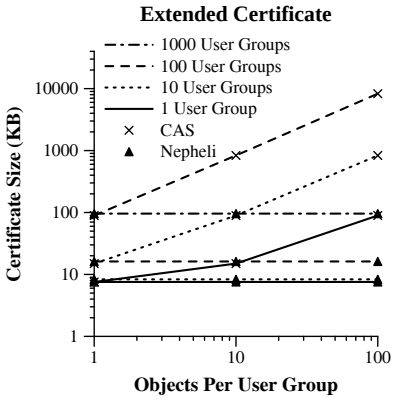
In this section, we summarize the qualitative differences of alternative architectural approaches to solve the problem of distributed access control. Then, we quantify the performance characteristics of the open-source implementation of the CAS service in comparison to the Nepheli prototype system that we designed and developed as main contribution of the present paper.

5.1 Qualitative Comparison

The CAS server represents one endpoint in the design space of distributed access control [16]. It uses a trusted third-party service to maintain the users, the objects and the respective allowed actions. Depending on the size of the federation, it is possible to require substantial administrative and computational cost, first to maintain the above data consistent with the corresponding entities of the participating domains, and second to access it prior to remote accesses across the federation. The fact that the CAS remains outside the administrative domain of the users leads to relatively slow connections for the remote communication with the CAS server over the Internet. The VOMS system recognized some of the above deficiencies and proposed to keep the action authorization function lo-



(a)



(b)

Figure 7: We consider extended certificates for a varying number of user groups and objects in each group, assuming 5 specified actions per object. We notice (a) that the CAS certificates for large numbers of groups and objects per group may take minutes to be issued by the server. Instead, the certificates issued by the Nepheli web service (a) only take few seconds even for one thousand groups per user. This behavior is consistent with the size of the generated extended certificate that we show in (b) for the CAS and Nepheli services.

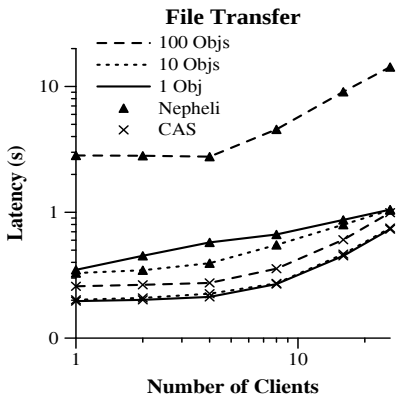


Figure 8: We measure the latency of using *url-copy* to transfer a single file of zero data size. We vary the number of accessible objects and the number of concurrent clients. Nepheli takes longer because it retrieves the policies online during the GridFTP request, while CAS has them embedded in the received certificate.

cal at each resource provider instead of having it centrally managed by the CAS server [1]. This approach releaves the central server from the need to account for all the shared objects and the allowed actions from each user. However, the VOMS server still centrally manages all the users and groups with remote access rights across the federation. Therefore, large federations may involve high overheads to continuously update the central VOMS server consistently with the users and their group memberships across the participating domains.

Another point in the design space of distributed access control has been introduced by the decentralized user authentication proposed for the Self-certifying File System (SFS) [9]. SFS manages the users and their groups locally at each participating domain, while each resource provider is responsible to authorize incoming remote requests based on locally cached name lists of the remote users and their groups. This approach successfully decentralizes the user authentication and action authorization functions across the federated domains. However, it introduces the need to have the authentication service of each user domain periodically transfer in-bulk the names of local users and groups (or their recent modifications) to the authorization service of the resource providers. In general, the cost of these updates is linearly dependent on the number of users that participate in each group.

Similarly, the Nepheli architecture also advocates the decentralization of the user authentication and action authorization services across the federated domains. However, we propose to have the access control lists of the resource providers specify access permissions with respect to federation-wide virtual groups. Then, the domain membership of each virtual group is updated periodically by the group's owner across the federation, while each remote access carries the list of virtual groups that have the requesting user as member. Correspondingly, the benefits of Nepheli include the local access of the authentication and authorization service at each domain along with the reduced update overhead required for the distribution of the domain membership of the virtual groups.

In the rest of the present section, we experimentally evaluate the cost to receive extended certificates from a web service depending on the size of the certificates and the type of attributes they carry. Subsequently, we quantify the cost of requesting file transfers from a GridFTP server with different types of certificates.

5.2 Extended Certificates

We measured the CAS server throughput and latency for a varying number of objects accessible by the requesting user. We used the Java-based command-line executable *cas-proxy-init* of the Globus Toolkit to request authorization certificates from the CAS server. We invoked the executable back-to-back repeatedly on a dedicated client machine in order to request certificates from the CAS and found the processor utilization on the client machine to exceed 35%. To minimize conflicts between concurrent requests, we used up to 26 different machines as clients to run a corresponding number of request loops to the same CAS server. In Figure 6(a), we use different curves to illustrate the throughput achieved to generate extended certificates for different numbers of accessible objects. It is evident, that throughput drops substantially as the number of accessible objects increases from tens to hundreds. Similarly, in Figure 6(b), we observe exponential increase in the delay to generate extended certificate of a thousand objects as we raise the number of concurrent clients.

However, the latency to generate an extended certificate even for one accessible object takes about 2s on an otherwise idle server. The corresponding utilization of the quad-core processors at the CAS server and database, respectively, increases up to about 50% for larger number of accessible objects. This implies that the processor cores are not fully utilized at the database engine and the Java virtual machine of the CAS web service, respectively. We used the debugging features of the Globus Toolkit to trace certificate requests through the web service and the database. At relatively low loads, we found that concurrently arriving requests would queue up at the server and handled sequentially one after the other instead of being served in parallel. We attribute this behavior to the scheduling of the Java threads by the Linux system on the multi-

Breakdown of Time (ms) to Issue Extended Certificate (Client - Server)

Method	*1 User Group*		*10 User Groups*		*100 User Groups*		*1000 User Groups*	
	CAS	*NPH*	*CAS*	*NPH*	*CAS*	*NPH*	*CAS*	*NPH*
getCasProxy() [Total client]	**1908**	**1840**	**2398**	**1809**	**6840**	**1879**	**98566**	**2459**
getCredential()	452	452	452	450	450	451	452	448
getSAMLAuthzQueries()	127	127	127	127	128	128	128	127
getCASPort()	356	359	357	357	354	356	376	354
getAssertion() [client]	596	562	764	526	3158	548	85497	708
getAssertion() [server]	**70**	**26**	**497**	**18**	**2091**	**25**	**82793**	**101**
parseAndVerifyAssertion()	154	134	331	137	1881	175	6162	414
embedAssertionInCredential()	187	171	331	176	830	186	5916	372
other	35	35	36	35	39	35	35	35

Table 2: We examine the time spent across different methods invoked at the client and server when we request extended certificates through the *cas-proxy-init* command. We compare the CAS (CAS) system against the Nepheli (NPH) prototype for varying number of user groups. Each user group is granted 5 action types over 10 different objects. The top row (*getCasProxy()[server]*) shows the total time to generate the extended certificate. In the case of CAS, the time spent at the server (*getAssertion()[Server]*) grows up to 83s, while in the case of NPH never exceeds 101ms. Overall, most of the time taken by the NPH system is spent at the client and the communication between the client and the server.

core hardware.

In Table 1, we measure the latency and throughput of the command *cas-enroll* that is provided by the Globus Toolkit to register new users into the CAS through the web service. When we ran multiple *cas-enroll* commands concurrently, the CAS server reported errors due to a bug that allowed a global class to be used at the same time by the independent commands. To overcome this limitation, we initiated multiple web services with different ports on the same server machine, all connected to the same database that run at a different machine. Then, we let the different concurrent clients each connect to a separate web service at the server machine. Following the above approach, we noticed that the request latencies remained almost flat across different numbers of clients, while the throughput increased almost linearly. In fact, when compared to the *cas-proxy-init*, the command *cas-enroll* is less heavyweight because it only carries the identity of one user, which results into relatively low communication delay and database access cost that remains constant across different requests. Nevertheless, every user enrollment takes around 1.69s, which could become a considerable cost for federated environments with numbers of users in the order of thousands, unless the system provides tools for in-bulk updates as was suggested for a different system by Kaminsky et al. [9].

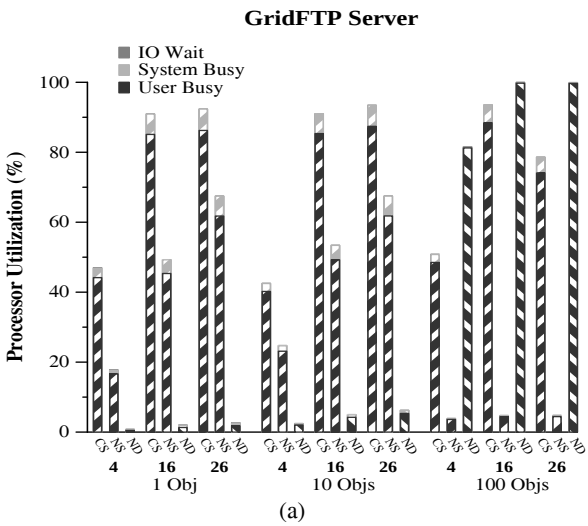
Subsequently, we considered the latency to generate extended certificates for a user that belongs to multiple user groups. We assume that each group gets five access permissions over each object for a varying number of objects. The certificate issued by the CAS server contains all the access policies that apply to the user. Instead, the certificate issued by the Nepheli system only contains the applicable virtual groups, that we emulate with normal CAS user groups. In Figure 7(a), we show that the time to issue a certificate takes up to minutes in the case of CAS for number of user groups and objects per group both equal to 100. On the contrary, the Nepheli system takes flat time to issue a certificate for varying numbers of objects. Furthermore, the latency remains less than 3s as we increase the groups that contain the requesting user from 1 to 1000. This behavior is consistent with the size of the issued certificate that we show in Figure 7(b) for the CAS and Nepheli systems. In fact, the CAS certificate grows up to about 10MB in the case of 100 user groups and 100 objects per group, while the Nepheli certificate does not exceed the size of 100KB even for 1000 user groups.

In Table 2, we show a breakdown of the time required to generate extended certificates for different numbers of user groups across the CAS (CAS) and the Nepheli (NPH) systems. The top row (*getCasProxy()*) corresponds to the main method invoked by the *cas-proxy-init* command at the client. Below that, we see the time spent across other important functions at the client, and the method *getAssertion()* called at the server. Across the different columns, we notice that most time is spent in the invocation of *getAssertion()* at the client and the server. In fact, the Nepheli system spends at the server about 10% of the total time, while the SOAP-based communication between the client and the server takes about a third of the total. We consider this observation about Nepheli crucial, because the generation of the certificate does not have to be done at a third-party web service as in the case of CAS (and VOMS). Indeed, we could have the user authentication service (UAS) of the Nepheli system located locally in the domain of each user. Thus, it is possible to have a more lightweight client-server communication in the case of Nepheli that will bring the total time to generate certificates much closer to the server time (*getAssertion()[server]*).

Overall, the generation of extended certificates by the Nepheli server only takes few seconds even for large numbers of user groups, unlike the CAS system where it takes up to minutes depending on the certificate size and the server load. Moreover, we anticipate this time in the case of Nepheli to be further reduced if the certification service runs locally in the domain of the user, where it can take advantage of more lightweight communication protocols.

5.3 File Requests

We experimented with the command-line executable *url-copy* that is provided by the Globus Toolkit to request file transfers from a GridFTP server. In the original CAS system, *url-copy* uses a CAS-issued extended certificate that carries the entire set or a subset of the access rights granted to the user through the specified administration policies. Then, the GridFTP server only extracts the applicable rights from the received extended certificate and matches them against the requested file and action. We also used the same command with Nepheli certificates that carry the virtual groups (emulated with normal user groups) that contain the requesting user as member. Essentially, the Nepheli system relies on the GridFTP server to extract the virtual groups from the certificate and subsequently retrieve the applicable policies from the local authorization database (Figures 2 and 5).



(a)

Figure 9: We measure the quad-core processor utilization on the CAS server (CS), the Nepheli server (NS) and the Nepheli database (ND). We vary the number of concurrent clients between 4, 16 and 26 along with the number of accessible objects between 1, 10 and 100. We have 5 allowed actions specified for each object. The Nepheli system has the database as bottleneck resource at 100 accessible objects and high client concurrency. Instead, the CAS system incurs higher utilization to the server machine.

We ran concurrent back-to-back *url-copy* requests on up to 26 different client machines against the same GridFTP server. We varied the number of accessible objects between 1 and 100. The requested file transfer is authorized by one of the accessible objects. Since our concern in this experiment is the cost of authorization, the requested file has zero data size. In Figure 8, we compare the latency of the *url-copy* between CAS and Nepheli for varying number of objects and concurrent clients. We observe that as the number of accessible objects increases, it affects the latency across both the systems. In the particular case of Nepheli with 100 accessible objects, the command takes up to 14.3 seconds to complete at high load. Instead, in the case of CAS, the transfer command never exceeds latency of 1s.

The above behavior is totally justified by the fact that Nepheli retrieves during the access request the applicable policies from the database, while the CAS system has the policies embedded in the received extended certificate. In comparison to Nepheli, CAS incurs higher database access load when it first generates the extended certificate, however, as we described previously for Figure 6(b). Indeed, we find it interesting that the certificate extension latency of the CAS server gets as high as 38s, which is more than twice as much in comparison with the extra delay that we observe in Figure 8 for the Nepheli-based GridFTP. In order to better understand the distribution of load across the system components, we also measured the processor utilization of the CAS GridFTP server, the Nepheli GridFTP server and the Nepheli authorization database (Figure 9). We observe, that the processor of the CAS GridFTP server is loaded up to 89% at large numbers of clients. Instead, in the case of the Nepheli system, the processor of the authorization database machine reaches utilization over 99% at 100 accessible objects and high client load, which clearly makes it the bottleneck resource.

5.4 Summary

From our experiments, we notice that embedding the access policies into the extended certificate issued by the CAS server can lead to high authorization delays at the CAS server. It is possible to choose to include a subset of the applicable policies into the certificate, but then the user has to retrieve more frequently different certificates depending on the particular files accessed each time. The Nepheli approach keeps constant the cost of issuing extended certificate regardless of the number of accessible objects. Correspondingly, Nepheli incurs extra delays to retrieve the policies online at the GridFTP server while serving the file transfer request. We notice that the added delay of the Nepheli system at the GridFTP server is lower in comparison to that at the CAS server. More importantly, the database access of the Nepheli system is amenable to additional optimizations, if we restrict the retrieved policies to those that match the known requested object, instead of having the database access being done only according to the specified user groups (Figure 5).

6. RELATED WORK

CRISIS is a wide-area authentication and access control system [3]. It uses the X.509 standard to define identity certificates for a principal, or transfer certificates to carry privileges from one principal to another. Roles associate a subset of a user's privileges with a name. Certificates are signed by a certification authority (CA) trusted by both endpoints of a communication channel. CAs sign all identity certificates with a long timeout and identify a local on-line agent to counter-sign the certificate with a shorter timeout. Unlike the Nepheli architecture, CRISIS requires the CAs to be organized hierarchically. Then, a principal believes a certificate endorsed by a foreign administrative domain if a path of trust exists from the local domain to the remote domain. Requesters are responsible to carry the necessary credentials that prove that they are authorized to access an object. Services have to maintain potentially large access control lists with principals and permitted operations for specific tasks.

Foster et al. proposed a general architecture to address security issues in distributed computational environments [4]. The user creates a certificate with temporary credentials to access remote resources, while resource providers at each organization map access requests from remote users to local accounts. The translation between global and local entities is done explicitly through a mapping table at the resource provider, or implicitly by having the user authenticated to both global and local authorities. In the case of large collaborations with many organizations and users, the maintenance of costly mappings across numerous entities poses scalability concerns.

A secure virtual enclave identifies a distributed collection of related resources along with the principals that are authorized to access them [20]. Each local administrator identifies the resources that may be shared, the principals that participate and the conditions that must be met before the access is allowed. A virtual enclave organizes resources into type equivalence classes, and principals into domain equivalence classes. The principal recognition rules assign principals to domains, and are propagated across collaborating enclaves. Client requests only carry identification and authentication data to an enclave, which subsequently authorizes the accesses of local resources based on principal recognition rules and local policies. The need for bulky transfers of principal recognition rules among the enclaves raises maintenance cost and policy concistency issues. The concept of virtual groups that we propose limits the

update overhead across different domains, because it involves the virtual group membership as list of entire domains instead of individual users.

Pearlman et al. point out that expressing policies in terms of direct trust relationships between producers and consumers across a community of organizations imposes flexibility and scalability restrictions [16]. They introduce a trusted third party called Community Authorization Service (CAS) to manage the policies that govern access to shared resources. The CAS server receives from the collaborating organizations the contributed users and resources, and organizes them into groups. It also supports policy statements that specify the allowed set of actions of a user group to a collection of resources. A user retrieves from the CAS server an extended X.509 certificate that specifies the permitted actions of the user over the registered resources. The operation of CAS as central trusted entity can become bottleneck in large installations because it involves continuous updates regarding the participating users, shared resources and allowed actions from all the organizations of the community.

The Virtual Organization Membership Service (VOMS) is a centralized system that manages the users of a virtual organization. The VOMS grants to users certificates with attributes that include their group membership, role assignment, and capabilities [1]. A resource provider uses the received attributes to allow the service of a request. VOMS only maintains a central server with the identities of the users and their groups, while keeping the resource configurations and access policies distributed across the resource providers. Thus, VOMS reduces considerably the server update overhead with respect to the CAS approach that used to store central information about both users and resources [16]. Nepheli uses virtual groups to decentralize the user authentication service across the participating domains, which is different from the VOMS system that maintains a central server to maintain all users and their groups in the federation [1].

Kaminsky et al. address the problem of user authentication in a global file system where specific users and groups from remote administrative domains are granted file service without any kind of preexisting administrative relationship [9]. The authors disregard support for remote principals with delegations through chains of certificates because such chains require additional infrastructure and complexity. Instead, users can specify at their local authentication server personal groups that contain remote users and groups. Authorization at the file server relies on access control lists with groups of local and remote users that are updated automatically through periodic communication among the authentication servers involved. The cost of group membership update between authentication servers is linearly dependent on the cardinality of the group. The overhead of caching remote group membership lists has also been independently evaluated in the context of grid computing [8].

The Grid Security Infrastructure (GSI) is a collection of functions for message protection, authentication, delegation and authorization [22]. Message protection can be provided through either TLS network connections or individually protected network messages. Users and services can be uniquely identified and asserted to another party using X.509 certificates. Proxy certificates allow bearers of X.509 end-entity certificates to temporarily delegate their privileges to another entity. The OASIS Security Assertion Markup Language (SAML) is an XML standard for exchanging authentication and authorization data between security domains. GridShib uses SAML to support attribute-based authorization in Globus-based grids and provides services based on certain user attributes associated with the client requests [19].

Zhang et al. propose an authorization management system for virtual organizations, which allows resource providers to define roles and virtual organizations to specify the users associated with a group [23]. If there is no virtual organization with pre-established authentication service, a resource provider creates separate groups and roles for each affiliated domain and allows the consumer domain to do the corresponding user assignment directly. Our approach is different because we reduce the authorization management overhead by having individual domain specify system-wide groups instead of different groups per domain and allowing each affiliated domain specify the users that participate in each group instead of having such assignment being done by centralized third-party services.

NFSv4 has been designed to overcome the limitations of previous distributed file systems for secure and efficient operation over wide-area networks [15]. The new protocol improves security by identifying users through domain-specific strings rather than numerical identifiers. Nepheli is different from the NFSv4, because it defines policies in terms of virtual groups rather than inidividual remote users. Leung et al. introduce the Maat system where they describe extended capabilities in order to authorize any number of clients to any number of files [11]. Extended capabilities keep their size fixed through appropriate hash representation of their contents. A comprehensive survey of decentralized access control in distributed file systems has been published recently by Miltchev et al. [13].

Keromytis and Smith examine the architectural requirements of scalable authorization services in large-scale networks [10]. Addressing the related system complexity requires mechanisms for policy specification, distribution and enforcement that handles large numbers of users and applications. An easy way to distribute policies to the enforcement points is through public-key credentials, which are public-key certificates that enclose authorization information. Administration complexity can be handled through multi-layer management approaches; the policies specified by different administrators are combined at a low or high level of the hierarchy and enforced at the evaluation point. Revocation of certificates is another important consideration in modern systems that has to be handled scalably.

7. CONCLUSIONS

We examined alternative approaches to apply distributed access control over federated file services, and concluded that existing solutions are hurdled by potentially heavyweight updates among independent administrative domains. We proposed the Nepheli architecture to keep the user authentication and action authorization services local at each domain, and introduced the virtual groups as two-layer user group structures that span multiple domains and only involve domain membership updates across different domains. We build a prototype Nepheli system and experimented with authentication and authorization requests for GridFTP transfers. In comparison to the CAS system, we found that moving the policy specification function to the resource provider, increases somewhat the latency of the GridFTP requests, but reduces substantially the cost to issue access certificates and makes it independent of the number of accessible object by a user. In our future work, we plan to extend the Nepheli architecture appropriately in order to apply it to distributed file systems for fine-grain accesses.

8. ACKNOWLEDGEMENTS

In part supported by project INTERSTORE with contract number I2101005 of the INTERREG IIIA Greece-Italy 2000-2006 program cofunded by the EU and the Greek State.

9. REFERENCES

[1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell' Agnello, A. Frohner, K. Lorentey, and F. Spataro. From gridmap-file to voms: managing authorization in a grid environment. *Future Generation Computer Systems*, 21:549–558, 2005.

[2] S. V. Anastasiadis, S. Gadde, and J. S. Chase. Scale and performance in semantic storage management of data grids. *International Journal on Digital Libraries*, 5(2):84–98, Apr. 2005.

[3] E. Belani, A. Vahdat, T. Anderson, and M. Dahlin. The crisis wide area security architecture. In *Usenix Security Symposium*, pages 15–30, San Antonio, TX, 1998.

[4] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communication Security*, pages 83–92, San Francisco, CA, Nov. 1998.

[5] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. Sharp: An architecture for secure resource peering. In *ACM Symposium on Operating Systems Principles*, pages 133–148, Bolton Landin, NY, Oct. 2003.

[6] The Globus Alliance. *GT 4.0.7 CAS Developer's Guide*.

[7] The Globus Alliance. *GT 4.0.7 GridFTP Developer's Guide*.

[8] J. Hemmes and D. Thain. Cacheable decentralized groups for grid resource access control. In *IEEE/ACM International Conference on Grid Computing*, pages 192–199, 2006.

[9] M. Kaminsky, G. Savvides, D. Mazieres, and M. F. Kaashoek. Decentralized user authenication in a global file system. In *ACM Symposium on Operating Systems Principles*, pages 60–73, Bolton Landing, NY, 2003.

[10] A. D. Keromytis and J. M. Smith. Requirements for scalabale access control and security management architectures. *ACM Transactions on Internet Technologies*, 7(2):1–22, May 2007.

[11] A. W. Leung, E. L. Miller, and S. Jones. Scalable security for petascale parallel file systems. In *ACM/IEEE Conference on Supercomputing (SC)*, 2007.

[12] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Wichel. Separating key management from file system security. In *ACM Symposium on Operating Systems Principles*, pages 124–139, Kiawah Island, SC, Dec. 1999.

[13] S. Miltchev, J. M. Smith, V. Prevelakis, A. Keromytis, and S. Ioannidis. Decentralized access control in distributed file sysems. *ACM Computing Surveys*, 40(3), Aug. 2008.

[14] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, Sept. 1994.

[15] B. Pawlowski, D. Noveck, D. Robinson, and R. Thurlow. The nfs version 4 protocol. In *International System Administration and Networking Conference*, Maastricht, Netherlands, 2000.

[16] L. Pearlman, V. Welch, I. Foster, and C. Kesselman. A community authorization service for group collaboration. In *IEEE Intl Workshop on Policies for Distributed Systems and Networks*, pages 50–59, Monterey, CA, 2002.

[17] N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, and T. Scavo. *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. Organization for the Advancement of Structured Information Standards (OASIS), Feb. 2007.

[18] R. S. Sandhu and E. J. Coyne. Role-based access control models. *Computer*, 29(2):38–47, Feb. 1996.

[19] T. Scavo and V. Welch. A grid authorization model for science gateways. In *International Workshop on Grid Computing Environments*, Reno, NV, Nov. 2007.

[20] D. Shands, R. Yee, J. Jacobs, and E. J. Sebes. Secure virtual enclaves: Supporting coalition use of distributed application technologies. *ACM Transactions on Information and System Security*, 4:103–133, May 2000.

[21] F. Taiani, M. Hiltunen, and R. Schlichting. The impact of web service integration on grid performance. In *IEEE Intl Symp on High Performance Distributed Computing*, pages 14–23, Research Triangle Park, NC, USA, 2005.

[22] V. Welch and the Globus Security Team. Globus toolkit version 4 grid security infrastructure: A standards perspective. Technical report, The Globus Alliance, 2005. Version 4.

[23] X. Zhang, Q. Li, J.-P. Seifert, and M. Xu. Flexible authorization with decentralized access control model for grid computing. In *IEEE High Assurance Systems Engineering Symposium*, pages 156–165, Nov. 2007.