Noname manuscript No.
(will be inserted by the editor)

Nephele: Scalable Access Control for Federated File Services

Giorgos Margaritis · Andromachi Hatzieleftheriou · Stergios V. Anastasiadis

Received: data / Accepted: date

Keywords Security · Access Control · Decentralization · File Services · Distributed Systems · Analytical Evaluation · Performance Measurements

Abstract The integration of storage resources across different administrative domains can serve as building block for the development of efficient collaboration environments. In order to improve application portability across such environments, we target data sharing facilities that securely span multiple domains at the filesystem rather than the application level. We introduce the *hypergroup* as an heterogeneous two-layer construct, where the upper layer consists of administrative domains and the lower layer of users from each participating domain. We use public keys to uniquely identify users and domains, but rely on credentials to securely bind users and domains with hypergroups. Each domain is responsible for authenticating its local users across the federation, and employs access control lists to specify the rights of individual users and hypergroups over local storage resources. In comparison to existing systems, we show both analytically and experimentally reduced transfer cost of remote authorizations and improved scalability properties.

1 Introduction

Different administrative domains frequently get involved in federations (or coalitions) that require secure data sharing among specific users [4, 7, 17, 30, 41]. Infrastructures that integrate storage resources from different domains can become the basis for flexible collaboration environments in several areas such as business, education, science and engineering. Different types of distributed computing already support secure resource peering among independent domains. Nevertheless, most existing applications are not designed to run across multiple machines, let alone machines that belong to different domains. Data sharing at the filesystem level has traditionally facilitated the interaction of standalone applications, but common distributed filesystems usually only run within a single domain.

G. Margaritis · A. Hatzieleftheriou · S. V. Anastasiadis
Department of Computer Science, University of Ioannina, Ioannina 45110, Greece

Today, secure file sharing across different domains remains a major challenge from several aspects including the scalable management of identities and access control [28, 30, 41, 50, 58, 61]. We are considering typical storage resources, such as the files on a fileserver, whose access control is mainly enforced by a common operating system. Role-based access control provides system administrators with a general intermediate construct to scalably assign access permissions to users in centralized and distributed computing systems [55]. Instead, ordinary operating systems usually apply discretionary access control, which allows users to specify resource permissions, for example, through the access matrix model. Alternatively, secure operating systems apply mandatory access control models, where only trusted administrators are allowed to specify the security policy enforced by the system [26].

We regard *security domain* (or simply domain) as an organization with independent security policy [39]. We assume that a *user domain* hosts users that consume resources, while a *resource domain* accommodates resource providers that handle access requests. In a federation, each domain often undertakes both the above functions. We set as a minimal requirement that each user domain maintains a local authentication service to identify the local users and their attributes, such as the groups that have a user as member. Similarly, every resource provider operates a local authorization service to control the permissions of requested accesses.

Traditional access control is effectively based on the identity of the requester, unless the user and resource lie in different domains. Then, a trusted third party (e.g. an X.509 certificate authority [38]) may certify the identity associated with a public key, while the resource provider securely maps the certified identity to specific permissions. Alternatively, a certificate-based approach has each request carry authorization assertions to the remote resource [60]. In an effort to reduce the administration cost, the Community Authorization Service (CAS) centrally gathers at a third-party server full information about the participating users and the accessible resources. Based on this information, the CAS server issues certificates of access permissions to the users [48]. In order to reduce the resource requirements of the central service, the Virtual Organization Membership Service (VOMS) distributes the management of access rights to the resource domains of the federation. However, VOMS centrally manages the identities of the individual users and the membership of the groups to which they belong [2].

Identity management is another challenge for secure cross-domain file sharing [41]. Due to the potentially complex authority infrastructure required by identity certificates, a public key (or its hash) can be directly used as a globally unique identity [14, 37]. Moreover, given the security risk of mapping identities to actions, distributed trust management uses *credentials* to directly bind public keys to actions [10]. Consequently, a fileserver may locally apply a user-agnostic default policy that grants access to user requests based on tamperproof permissions carried by the requests [40].

Despite the availability of several access control models, the access control list (ACL) remains the main protection mechanism for general-purpose fileservers [26]. Since fileservers manage file permissions locally, they could also handle remote requests through local ACLs, if the ACLs directly contain users or user groups from remote domains [30]. The assignment of access permissions to user groups provides a relatively scalable mechanism to uniformly specify and enforce access control for large numbers of users that span multiple domains [41]. Furthermore, the Self-certifying File System (SFS) periodically replicates group memberships between the authentication and the authorization service of different domains in order to avoid complex certification infrastructures and the need of users to attach attributes (e.g. user groups) to their requests [30].

In a large federation consisting of numerous domains and users, there are potentially several large groups each consisting of multiple subgroup layers across the different domains [46,50]. Therefore, depending on the modification frequency of group memberships, cross-domain replication may incur sensitivity to network transfers and outages. The presumed need for advance propagation of group memberships to a fileserver is based on the assumption that request-carried certificates should be avoided due to the complexity of the certificate chains that they introduce [7,9,30].

In the design range of access control for distributed file services, full support or complete avoidance of certificates is two (potentially costly) extremes. As a compromise between the two, we propose to specify ACL permissions in terms of an heterogeneous two-layer construct that we call *hypergroup*. The upper layer is a group of domains, while the lower layer is the union of user groups contributed respectively by the domains of the upper layer. Then, a remote access request to a fileserver only carries a credential (signed attribute) of the hypergroups to which the requesting user belongs. Correspondingly, the credential only contains hypergroup identities rather than full access permissions, and the fileserver only needs to know the hypergroups that contain a particular user rather than the full user membership of each hypergroup.

The heterogeneous structure clearly differentiates hypergroups from traditional constructs, such as roles and groups. In fact, a domain dynamically creates a hypergroup and selects the domains of a federation that contribute users or fileservers. Thus, we can apply access control without central management of the users or their access rights. The respective certification cost is low for several reasons:

1. The proposed credential only certifies which hypergroups contain a user and a requested fileserver, rather than all the hypergroups in the federation.
2. The credential only contains hypergroup identities rather than the full user population of each hypergroup. For example, a fileserver may not need to know those particular remote users that will never access it.
3. A user domain never needs to disclose externally the entire hierarchy of local groups that potentially participate in a hypergroup. Actually, a domain determines the local users that belong to a hypergroup based on the local administration practices independently of the other domains [45]. As a result, we facilitate administrative autonomy and privacy across the federated domains.

We achieve improved scalability with respect to the CAS centralized approach because we manage the user identities and access permissions locally at the respective user domains and resource domains. In comparison to VOMS, we additionally avoid the centralized management of user identities because we specify the upper layer of a hypergroup in terms of domains rather than individual users. Similarly, we avoid the network traffic that SFS requires to propagate group memberships specified in terms of users; we only propagate across the federation the hypergroup memberships specified in terms of domains. We further explain the above difference when we compare analytically and experimentally our system with alternative architectures in Sections 6 and 7.

To the best of our knowledge, the present work is the first to propose the hypergroup as an heterogeneous multi-layer construct with flexible domain membership to provide scalable decentralized authorization in federated environments. Additionally, we help fill an important gap in the published literature through the quantitative comparison of alternative authorization models that we conduct. In summary, the main contributions of the present work include

- Reconsideration of the security requirements for scalable access control in federated file services.
- Description of a security model based on the construct of hypergroups that we introduce.
- Prototype implementation of the Nephele system within the Globus Toolkit standard middleware environment.
- Analytical comparison and experimental evaluation of the performance properties of the Nephele prototype with respect to other systems.

In the rest of the manuscript, we describe our trust model in Section 2, and introduce hypergroups in Section 3. In Section 4 we present the architecture of Nephele, while in Section 5 we get into details of our prototype implementation. In Section 6 we analytically compare the storage and transfer requirements of our architecture with those of other systems. We outline our experimentation environment in Section 7, and experimentally compare the Nephele prototype system with CAS. We present previous related work at Section 8, and summarize our conclusions in Section 9.

2 Goals and Trust Model

We set as research objective to enable collaborations over the Internet through secure file sharing with low overhead. For several reasons that have to do with security concerns and transfer overheads, block-based accesses remain challenging over wide-area networks in comparison to large file downloads [47]. As faster backbone networks blur this distinction, we mainly target fine-granularity accesses over large distances. For that purpose, we set the following goals:

1. *Federation support.* We facilitate collaboration by allowing domains to dynamically organize federations of file services over the network.
2. *Cross-domain operation.* We enable the users of one domain to securely access the fileservers of a remote domain according to the principle of least privilege [52].
3. *Fileserver compatibility.* For compatibility with common fileservers, we use access-control lists to manage the access permissions at file and user granularity.
4. *Decentralized administration.* We distribute the security state across multiple domains to avoid centralized management of the federation.
5. *Communication scalability.* We require low communication cost to control accesses within and across the domains that participate in the federation.

In order to establish mutual trust, the collaborating domains exchange public keys through a secure external registry. For instance, domain public keys may be

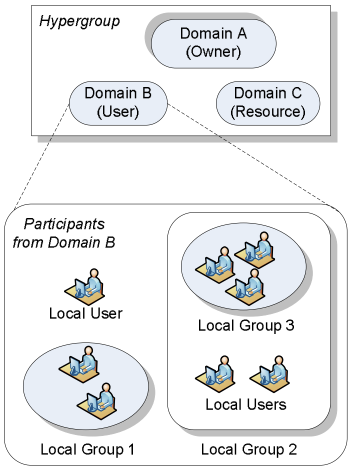


Fig. 1 Each participating domain contributes to the hypergroup an arbitrary hierarchy of local users and groups.

distributed via a global registry such as the Security Extensions (DNSSEC) of the Domain Name Service [19]. However, we leave key distribution outside the scope of the present work because it is studied as an independent problem elsewhere [37]. We deliberately avoid complex multi-level authentication among the domains, or long delegation chains across users. Even though academic communities and social networks commonly allow individual users to easily share data with remote untrusted users, there are also typical environments (e.g. commercial) where such flexibility is restricted for security reasons. Accordingly, we regard as acceptable the involvement of security administrators in the configuration of a federation without compromising administration scalability.

Multiple domains directly establish a federation through out-of-band negotiation. We rely on public keys (or hashes thereof) to uniquely identify the participating users and services. We trust the federated domains to correctly identify the participating local users, while we have the fileservers manage the respective access permissions. Remote accesses occur over an untrusted network that requires symmetric encryption to ensure the confidentiality (and integrity) of the exchanged data. Secret shared keys can be established online over the public keys of the corresponding endpoints [30]. For efficiency, we may cache and reuse secure network channels to amortize the cost of the initial asymmetric cryptography involved.

We enable remote fileservers to securely determine the applicable access permissions based on user credentials that we embed into access requests. However, we limit credentials to simply contain signed identity attributes of the users rather than policy statements with permitted actions over the requested file resources. Thus, our model combines the goals of decentralized administration and fileserver ACL compatibility.

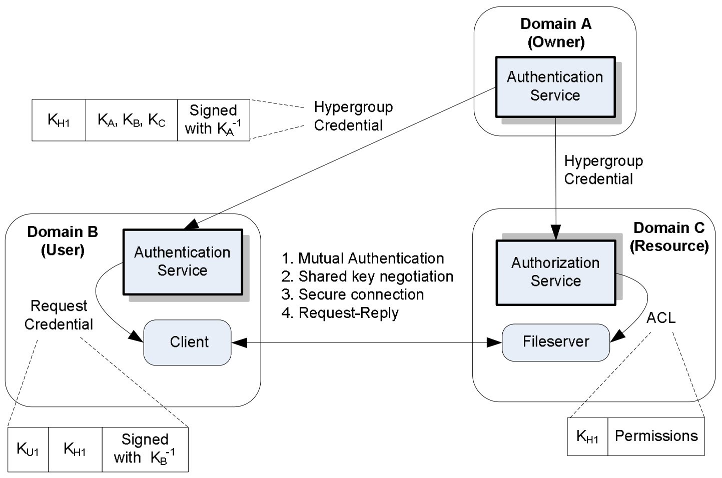


Fig. 2 The Nephele architecture keeps the identity authentication of the users separate from the authorization of their requested actions at the server. The owner domain A signs the credential of hypergroup H_1 with the private key K_A^{-1}, while the user domain B signs the request credential of user U_1 with the private key K_B^{-1}.

3 Hypergroups

The hypergroup is an abstract construct that we introduce to enable flexible and secure file sharing between different domains. The hypergroup is semantically similar to the group of traditional operating systems, but it has a two-layer hierarchical structure that spans multiple domains. The upper layer consists of domains, while the lower layer contains the users that participate from each domain.

The administrator of a domain can dynamically create and globally identify a hypergroup as a distinct public key. Correspondingly, the creating domain is called the *owner* of the hypergroup and bears the responsibility to maintain the set of domains that are members of the hypergroup. For simplicity, we assume that the owner automatically becomes the first member of the hypergroup. Subsequently, each member domain is responsible to specify the local users that belong to the hypergroup (Fig. 1). Additionally, we have as members of the hypergroups those domains that simply contribute file resources rather than users. This makes relatively straightforward to find those hypergroups that contain the domains of both the user and the accessed resource in a request.

Example. The administrators of the participating domains communicate offline to agree on the hypergroups that will be needed in a federation and the domain that undertakes the ownership of each hypergroup. Practically, the number of hypergroups depends on the distinct types of users that participate in the federation. We consider the hypothetical joint project *fedstore* which is established between the domains A, B and C. Let K_D denote the unique public key that globally identifies entity D, and K_D^{-1} the respective private key (Fig. 2). If the domain A is the project coordinator, it may own the hypergroup $K_{admins@fedstore}$ for the project administrators, $K_{scientists@fedstore}$ for the research staff, and $K_{engineers@fedstore}$ for the engineering personnel. Each of the three hypergroups is defined as a list of three domains $\{K_A,\ K_B,\ K_C\}$, and each of the three domains specifies the local users that belong to each hypergroup.

4 System Design

In the present section, we describe important parts of the Nephele architecture related to the functions of authentication, authorization, administration, delegation, revocation, and accountability.

Authentication. Domains, users, hypergroups, and services are global entities uniquely identified through their public keys. Thus, a user and a fileserver can mutually authenticate to each other, and subsequently establish a secure connection for request processing. Each domain operates a local authentication service that issues *credentials* as signed bindings of public keys with specific identity attributes. In particular, the domain that owns a hypergroup issues a *hypergroup credential* to enlist the domains that belong to the hypergroup.

We consider *home domain* of a user the domain that registers the user as local entity. Accordingly, the home domain of a user issues a *user credential* to enlist the hypergroups that contain the user. For reduced communication cost, the home domain may issue a *request credential* that only enumerates those hypergroups which contain both the domains of the user and the fileserver specified in a remote access request (Fig. 2). We envision filesystem clients that automatically obtain request credentials without the manual intervention of the user. As a defensive measure against attacks, credentials have a limited lifetime at the end of which they become invalid unless they are renewed.

Authorization. Each fileserver specifies the action permissions of local or remote users and hypergroups. For that purpose, ACLs are commonly used by fileservers due to the fine granularity, administration flexibility and user accountability that they offer. By placing both policy specification and enforcement within a resource domain, we keep low the amount of inter-domain state transfer. Hypergroup owners are responsible to periodically inform the current members of the hypergroup about the respective domain membership. We anticipate this update to be relatively inexpensive given that the number of domains that belong to a hypergroup should typically not exceed a few tens or hundreds at most. Instead, the population of users that participate in a hypergroup can be large and change frequently, which would make the corresponding inter-domain update substantially more costly.

Nephele has each remote access carry a request credential issued by the home domain of the respective user. The source network address along with the current time (or a nonce) can be securely attached to the exchanged messages to strengthen their authenticity and freshness [44]. The remote fileserver executes the access, if the user or his hypergroup is granted the respective permission in the ACL of the requested file. Thus, the authorization decision is decentralized across the three domains involved in an access: (i) The user domain that issues the request credential, (ii) The hypergroup owner that issues the hypergroup credential, and (iii) The resource domain that hosts the fileserver.

Administration. Each domain retains the management of the respective entities –local users, hypergroups or fileservers– that it maintains. The request credential issued by a domain only exposes to other domains the hypergroups that contain a requesting user. As we already explained, each domain applies whatever grouping method fits better to identify the users that participate in each hypergroup, e.g. the role hierarchy or the organizational structure. Consequently, Nephele improves administrative transparency and efficiency because it makes unnecessary for a domain to disclose (and update) its internal structure to other domains. The selective participation of domains into hypergroups directly supports the principle of least privilege [52], while the ability of each domain to create hypergroups naturally decentralizes the federation administration. A local security policy may allow individual users to create multi-domain hypergroups without the involvement of an administrator. Nevertheless, the creation of hypergroups and the corresponding user assignment by administrators should adequately handle the maintenance needs of a typical collaboration.

Delegation. Delegation of access rights across a distributed system is either a means to execute jobs that involve multiple networked servers with different access restrictions, or a collaboration mechanism that allows a user to enable resource sharing with users from remote domains [7, 29, 62]. In Nephele, we grant local access rights to a remote user by first making a file accessible to a hypergroup, then adding the home domain of the user to the hypergroup, and finally having the remote user added to the hypergroup by their local administrator. Therefore, for the security needs of remote file sharing, we support some limited form of inter-domain delegation. In particular, our scheme assigns to the hypergroup owner the specification right of the hypergroup domain membership, while it assigns to each participating domain the specification right of local user membership in the hypergroup. Nevertheless, we don't expect this simple delegation scheme to entail the deployment concerns previously expressed for complex certificate chains [30].

Revocation. Our model inexpensively provides revocation of access rights through the time expiration of hypergroup and request credentials. This mechanism implicitly notifies a fileserver that an individual user or an entire domain no longer belongs to a hypergroup, which automatically invalidates the applicability of the hypergroup access rights to the user or domain. As an option, we also enforce immediate cancellation of rights through the explicit distribution of revocation lists from a user domain or a hypergroup owner to a resource domain. Such a solution provides immediate policy enforcement, but increases the system cost and complexity with the extra updates required [7, 23].

Accountability. We make a user accountable for his requests through a mutual authentication step required for the initial connection with the fileserver. Subsequently, each request carries a request credential signed by the respective home domain. Thus, we can audit the remote resource consumption by each user, and trace back system misbehaviors to the user and domain that caused them. It is possible that a compromised domain falsely issues request credentials to unauthorized local users, or a hypergroup owner falsely adds unauthorized domains to the federation. Nevertheless, a user or domain is unable to impersonate a different entity undetectably, provided that public keys are securely distributed across the federation. Ultimately, a misbehaving domain can be easily removed from a hypergroup, and similarly the hypergroups of a malfunctioning owner can be entirely deleted from the fileservers in the federation.

Summary. We can outline the advantages of our approach as follows:

1. *Decentralization.* We avoid the requirement from any authority to know all users, hypergroups or accessible resources.
2. *Freshness.* User population changes are configurably reflected into user requests according to the frequency that request credentials are refreshed.
3. *Access Cost.* Requests only need to carry request credentials that simultaneously contain both the domains of the user and the fileserver.
4. *Update Cost.* We reduce the update overhead at the fileservers by only requiring them to know the domain membership of the hypergroups.

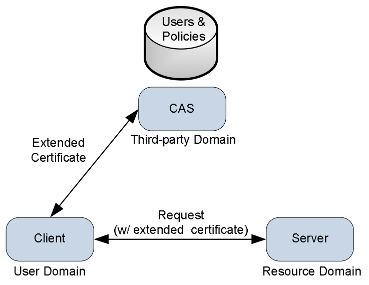
5 Prototype Implementation

In order to evaluate the authorization cost of hypergroups in comparison to traditional certificates, we built a prototype of the Nephele architecture based on the open source implementation of the Community Authorization Service (CAS) [48]. Next, we describe the modifications that we applied to CAS and GridFTP in order to develop a prototype of the Nephele architecture.

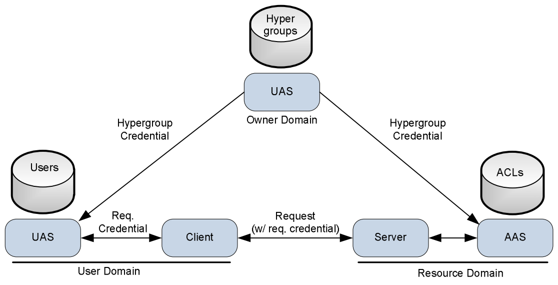
The original CAS server includes a front-end SOAP-accessible web service written in the Web Service Definition Language, a back-end commodity relational database accessed through embedded SQL statements, and Java code that links these two components [20, 48]. The relational database centrally manages a collection of tables that keep track of users, resources, and service actions, as shown in Fig. 3(a). 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 [21]. 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 access rights that apply to the user. The GridFTP server receives the certificate from the client and applies the embedded access permissions to decide if the requested service action is allowed for the identified user. A prior CAS profiling study attributes a substantial part of the certification cost to the processing needed by the web services and the encryption library [59].

The CAS architecture simplifies the trust infrastructure because it establishes mutual trust relationships between the CAS server and each domain, instead of among all the domains between each other. With hypergroups, we aim to avoid single points of failure and also have simple trust relationships between each hypergroup owner and the user or resource domains. 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) (Fig. 3(b)). 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 hypergroups. Correspondingly, each resource provider operates its own AAS to manage the access control. Thus, Nephele relocates the definition of user groups from the central CAS server to the UAS of each domain and the specification of access control from the central CAS server to the AAS of each fileserver.

The Nephele prototype issues to the user a request credential that is based on X.509 certificate, extended with the Security Assertion Markup Language (SAML) [51]. We implemented the UAS subsystem of Nephele in Java code, which



(a) Original CAS



(b) Nephele prototype

Fig. 3 (a) The original CAS server centrally maintains the users, objects, actions and their respective groups along with the permitted actions of the users over the objects. (b) The Nephele prototype places the user attributes at the User Authentication Service (UAS) of the user domain, and the ACLs at the Action Authorization Service (AAS) of the resource domain. We maintain the hypergroups at the UAS of each owning domain.

retrieves from a database the groups that contain the user directly and iteratively accumulates the groups that contain the user indirectly. The GridFTP client forwards the credential along with the file transfer request to the GridFTP server. During the file request processing, the server parses the SAML extension to extract the hypergroups of the user and retrieves the applicable access rights from the local AAS. The above functionality was partially available in the original CAS implementation across approximately 9K lines of Java code. However, we had to port it from the Java-based CAS server 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 a library (e.g. 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. We disregarded the manual code translation due to the effort replication that it involved, and the direct use of Java code from C due to skepticism about the performance overhead that it would introduce. Instead, we decided to implement the required database access of the AAC with a single database statement that would both select the relevant tuples and join them into the required list of fields at one step. We temporarily store the returned permissions into a list of objects and allowed actions. Then, the server

Asymptotic Costs

System Architecture	*Central Storage (Num. stored items)*	*Periodic Updates (Num. item transfers)*	*Certif. or Credent. Size (Num. carried items)*
CAS	$O((M+G)\cdot N)$	0	$O(M)$
VOMS	$O(G\cdot N)$	0	$O(G)$
SFS	0	$O(D\cdot G\cdot N)$	0
Nephele	0	$O(D^2\cdot H)$	$O(H)$

Table 1 Across four representative architectures, we summarize the number of items involved in central storage, periodic updates and remote access certificates or credentials. We assume that $D \ll N$ and $H \leq G$.

matches the list against the requested file and operation. If the match succeeds, the server proceeds to the execution of the requested operation.

Our prototype is relatively unoptimized, since a typical fileserver implements the access-control functionality internally instead of assigning it to an external database. In a real deployment, the client and server domains are also probably connected over remote links. Then, the request credential is issued by the UAS at the home domain of the user, while the authorization request is processed by the AAS that resides at the domain of the fileserver. Therefore, in the common case, the only interaction needed over the wide-area network is the submission of the file transfer request to the remote server along with the file transfer itself. Consequently, in our experimental measurements we focus on the cost of having the request credential of a user issued by the local UAS, and the cost of a file transfer request served by the GridFTP server in collaboration with the local AAS.

6 Analysis and Qualitative Comparison

We consider alternative representative architectures to solve the problem of distributed access control. In the present section, we point out several qualitative differences among the architectures, and analytically approximate the number of items that is maintained at a central server or transferred among the domains. We assume a total number of N users, H hypergroups, G user groups (or roles) and M resources across a federation of D domains. Based on the system description of Section 4, it is reasonable to claim that $D \ll N$ and $H \leq G$.

The Community Authorization Service (CAS) represents one endpoint in the design space of distributed access control [48]. It uses a trusted third-party service to maintain the identities and properties of users, groups, resources, and allowed actions per resource and user. The CAS server centrally stores a total of $O(N+G+G\cdot N+M+M\cdot(N+G)) \approx O((M+G)\cdot N)$ entities assuming that $G \ll N$. Since the server lies outside the user domains, the CAS is accessible over the Internet through relatively slow connections and protocols. For remote accesses, the user needs a CAS-issued extended certificate of size $O(M)$ to specify his permissions over specific resources. In a large federation, the CAS is likely to require substantial overhead to keep the stored information up to date, and issue certificates prior to user remote accesses [27].

In contrast, the Virtual Organization Membership Service (VOMS) keeps the authorization function local at each resource provider [2]. Thus, it removes from the central server the need to account for the M shared resources and the respective actions allowed to each user. However, VOMS remains responsible to centrally manage the N users and their G groups (or roles) in the federation. For simplicity, with the symbol G we refer to both groups and roles of VOMS. Asymptotically the number of centrally stored entities is equal to $O(N + G + G \cdot N) \approx O(G \cdot N)$. A remote access carries a certificate of size $O(G)$ enlisting the relevant groups (or roles) of the respective user. Consequently, VOMS is also likely to incur high overheads for the continuous update of the centrally maintained entities and the respective issuance of certificates by the central server.

The decentralized user authentication of the Self-certifying File System (SFS) is another unique point in the related design space [30]. It manages the N users and their G groups locally at each participating domain, and each resource provider authorizes incoming remote requests based on the locally cached lists of the remote users and their groups. Thus, SFS decentralizes the functions of user authentication and action authorization across the federated domains, but the authentication service of each user domain has to periodically transfer the updated user groups to the authorization service of the resource. Essentially, each user domain transfers $O(N + G + G \cdot N)$ items to all the resource domains, which roughly amounts to a total of $O(D \cdot (N + G + G \cdot N)) \approx O(D \cdot G \cdot N)$ item transfers. Practically, it is only necessary to transfer the group modifications that occurred over time, which substantially reduces the amount of transferred data. The authorization of a remote access is done at the resource domain on the basis of the (globally unique) user identity carried by the request.

The Nephele architecture also favors the decentralization of user authentication and action authorization across the domains. However, the access control lists of the resource providers specify the permissions in terms of the H hypergroups, with each hypergroup containing up to D domains. Thus, the periodic update of the H hypergroups across the federation involves the transfer of $O(H + H \cdot D)$ items from each home domain to the remaining domains. This cost roughly amounts to a total of $O(D \cdot (H + H \cdot D)) \approx O(D^2 \cdot H)$ item transfers. As with the groups of SFS, it is only necessary to update hypergroup modifications over time. Consequently, each remote access carries a list of the $O(H)$ hypergroups that have the requesting user as member.

We summarize the storage and transfer costs of the four architectures in Table 1. In comparison to CAS and VOMS, both SFS and Nephele completely avoid the centralized management of users or resources. Unlike SFS, Nephele additionally avoids to transfer user lists from one domain to another. Thus, the authentication and authorization services of Nephele only involve local accesses at each domain, while the hypergroup updates across the federation contain a small amount of data that specify domain lists.

7 Performance Evaluation

We quantify the performance characteristics of the CAS service in comparison to the Nephele prototype that we designed and developed. In our experiments, we measure the latency to issue Nephele request credentials and authorize file transfers with respect to the corresponding costs of the original CAS and GridFTP server. Additionally, we measure the latency to insert users into a CAS server so that we quantify the potential cost of centralized user management. Hypergroup credentials have similar structure to request credentials (Fig. 2), and we do not evaluate them separately.

7.1 Experimental Environment

For our measurements, we used an isolated cluster of 28 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, up to 3GB RAM, one enabled 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 (GT) 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 use GT in our comparative experiments because it already supports certificates and is modular enough to allow the placement of the authentication and authorization functions at different domains. Nevertheless, prototyping Nephele over GT was a challenging task that took us several months due to the lack of unified, detailed documentation about the internal organization of the multithreaded modules that we modified.

We used one machine to exclusively run the database, and another to execute in turn the CAS, the Nephele or the Gridftp service. In order to concurrently handle the workload requests, we set equal to 50 the maximum number of threads in the CAS web service, and equal to 100 the number of database connections to the web service. We generate the client requests against the above services with up to 26 separate machines, which make possible to include both low and high loads in our study. Essentially, our experiments only involve networking and memory-based processing on otherwise idle machines. After we ensured that our measurements had negligible variation across different experiment repetitions, we show averages for the handling of 50 up to 200 requests. In each experiment, we only take measurements when all concurrent clients become active, after an initial warm-up period. We also desynchronize the concurrent client requests by starting the clients one after the other with brief random delays between them.

7.2 Certificates and Credentials

We measured the CAS server latency and throughput to issue extended certificates for a varying number of objects accessible by the requesting user. We use the Java-based *cas-proxy-init* command of the Globus Toolkit (GT). We invoke the executable back-to-back repeatedly on a dedicated client machine in order to request certificates from the CAS. This load raises the processor utilization on the client machine above 35%. To avoid client competition between concurrent requests, we used up to 26 different client machines and ran a corresponding number of request loops against the same CAS server. In Figure 4(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.

In Figure 4(b), we observe exponential increase in the delay to generate extended certificate of a thousand objects as we raise the number of concurrent

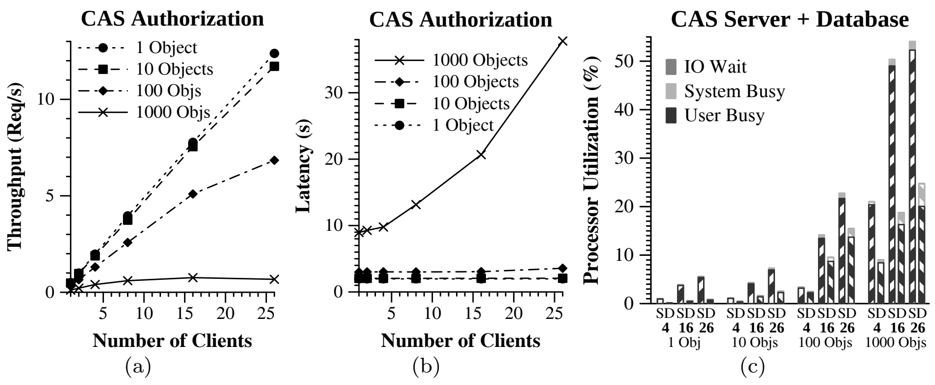


Fig. 4 We examine the performance of a CAS server that issues extended authorization certificates for number of concurrent clients between 1 and 26. We grant five action rights to each object. For certificates with large number of objects (e.g. 1000), we observe the server throughput (a) to drop substantially and the request latency (b) to reach 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 shown for 4, 16 and 26 clients.

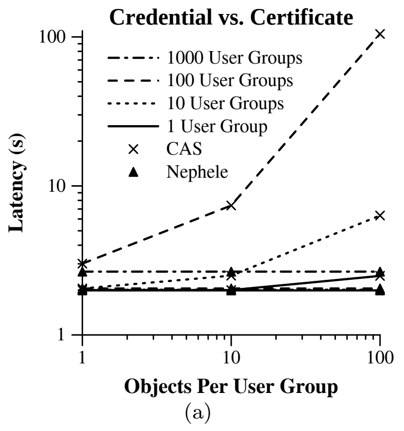
User Enrollment into CAS

# Clients	*Latency (s)*	*Throughput (Rq/s)*
1	1.6906	0.5882
2	1.6922	1.1628
4	1.6802	2.2989
8	1.6876	4.5455
16	1.6984	8.6957
26	1.7275	13.2653

Table 2 We enroll new users through multiple concurrent container processes at the CAS server. We observe that latency remains almost flat and throughput scales approximately linearly. Nevertheless, registering new users to a third-party entity through the web service turns out to be a relatively time-consuming procedure.

clients. The latency to generate an extended certificate even for one accessible object takes about 2s on an otherwise idle server. Despite the high load, the respective processor utilizations at the CAS server and database do not exceed 55% (Fig. 4(c)). This implies that the processor cores are not fully utilized by the database engine and the CAS web service. We used the debugging features of the Globus Toolkit to trace the generation of certificate requests through the web service and the database. Even at low loads, we found that concurrently arriving requests queue up at the server to be handled sequentially one after the other instead of being served in parallel. We attribute this behavior to the interaction of the CAS system with the scheduling of the Java threads by the Linux kernel.

In Table 2, we measure the latency and throughput of the *cas-enroll* command included in the Globus Toolkit to register new users into the CAS server. When we ran multiple commands concurrently, the CAS server reported errors. We traced this problem to a feature (potential bug) of the original code that allowed a global class to be used at the same time by concurrent requests. To overcome this limitation, we initiated multiple web services with different ports on the same server



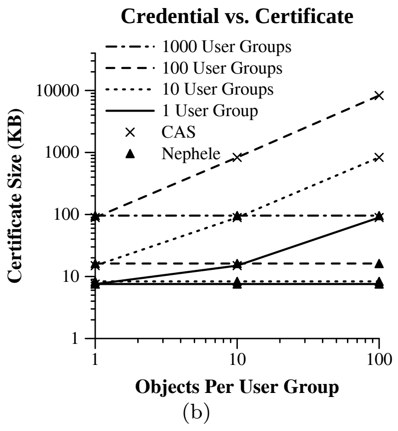


Fig. 5 We consider groups with a varying number of objects and fixed number (five) of allowed actions per object. (a) For large numbers of groups and objects, the CAS certificates may take minutes to be issued by the server. Instead, the Nephele credentials only take few seconds even for 1000 groups per user. (b) This behavior is also reflected in the size of the generated certificate and credential shown for CAS and Nephele. The case of 1000 User Groups is only shown for the Nephele system due to the excessive size and latency that it incurs in CAS.

machine, all connected to the same database that ran at a different machine. Then, we let the different concurrent clients each connect to a separate web service at the server machine. In our measurements, we found the request latencies approximately flat across different numbers of clients, while the throughput increased almost linearly. When compared to the *cas-proxy-init*, the command *cas-enroll* is less heavyweight because it only carries the identity of one user; this results into relatively low communication delay and database access cost that remains constant across different requests. Nevertheless, every user enrollment takes around 1.69s; this 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 [30].

Subsequently, we considered the latency to generate extended certificates for a user that belongs to multiple user groups. We assume that each group gets a fixed number of access permissions per object for a varying number of objects. We set the number of permissions equal to five (e.g. read, write, execute, sticky and setuid permissions of Unix). The certificate issued by the CAS server contains all the access policies that apply to the user. Instead, the certificate issued by the Nephele system only contains the applicable hypergroups emulated with normal CAS user groups. In Figure 5(a), we show that the time to issue a certificate takes up to minutes in the case of CAS when we set both the number of user groups and objects per group equal to 100. On the contrary, the Nephele system takes flat time to issue a credential 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 appears in Figure 5(b). The CAS certificate grows up to about 10MB in the case of 100 user groups and objects per group, while the Nephele credential does not exceed the size of 100KB even for 1000 user groups.

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

	1 Group		*10 Groups*		*100 Groups*		*1000 Groups*	
Method	*CAS*	*NPH*	*CAS*	*NPH*	*CAS*	*NPH*	*CAS*	*NPH*
getCasProxy() [Client Total]	**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 3 We examine the time spent across different methods of the client and server, when we request certificates or credentials with the *cas-proxy-init* command. We compare the CAS (**CAS**) and Nephele (**NPH**) systems for a 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. The time spent at the CAS server (*getAssertion()[Server]*) grows up to 83s, while it reaches 101ms in the case of Nephele. It is notable that most of the Nephele time is spent at the client and the communication between the client and the server.

In Table 3, we show a breakdown of the time required to generate extended certificates for different numbers of user groups across the CAS and the Nephele systems. The top row (*getCasProxy()*) corresponds to the main method invoked by the *cas-proxy-init* command at the client. Beneath 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, respectively. In fact, the Nephele system only spends at the server about 10% of the total time. Instead, the SOAP-based communication between the client and the server takes about a third of the total time. We consider crucial this observation about Nephele, because the generation of the certificate does not have to be done at a third-party as in the case of CAS (or VOMS). Indeed, we suggest having the user authentication service (UAS) of the Nephele system located locally at the domain of each user. Thus, a more lightweight client-server communication in the case of Nephele could bring the total time to generate certificates much closer to the server time (*getAssertion()[Server]*). Overall, the generation of credentials by the Nephele 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.

7.3 File Requests

We experimented with the *url-copy* command of the Globus Toolkit to request file transfers from a GridFTP server. In the original CAS system, *url-copy* uses an extended certificate that carries the set of access rights granted to the user. Then, the GridFTP server extracts the applicable rights from the received extended certificate and matches them against the requested file and action. We also ran the command with Nephele credentials that only carry the hypergroups of the requesting user. The Nephele system relies on a modified GridFTP server to extract the hypergroups from the credential and subsequently retrieve the applicable policies from the server authorization database.

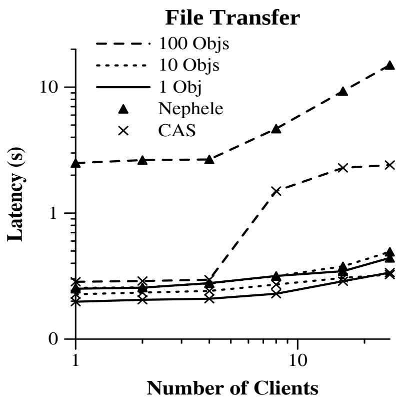
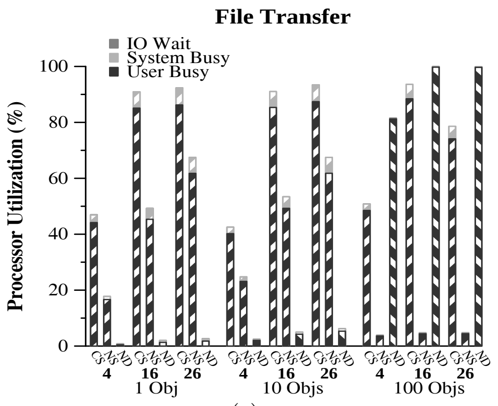


Fig. 6 The Nephele GridFTP server takes longer to complete the file transfer because it retrieves the policies online unlike the CAS system that has them embedded in the certificate carried by the request. At 100 accessible objects, latency in both systems increases substantially for ≥8 clients.



(a)

Fig. 7 We measure the quad-core processor utilization on the CAS GridFTP server (**CS**), the Nephele GridFTP server (**NS**) and the Nephele database (**ND**). At 100 accessible objects and 16 clients or more, the system is saturated in both cases. The bottleneck resource is located at the database machine for Nephele, and the GridFTP server in the case of CAS.

In order to evaluate how the file service is affected by the certificate type, we ran back-to-back requests concurrently from up to 26 client machines against the same GridFTP server. We vary the number of accessible objects between 1, 10 and 100, while we authorize the requested file transfer to one of the objects. Since we focus on the authorization cost, we request files with zero data size. In Figure 6, we compare the latency of the *url-copy* command over CAS and Nephele. For 1 or 10 accessible objects, increasing the number of concurrent clients leads to comparable latencies across the two systems. In additional measurements for 1 or 10 objects, we also found the CAS GridFTP server to operate at processor utilization that exceeds 92% for 16 or 26 clients; the total utilization of the Nephele GridFTP server and the database remains up to 77% across the different numbers of clients (Fig. 7).

Instead, at 100 accessible objects, the file transfer takes 2.5-14.96s with Nephele and only 0.29-2.4s for CAS. We previously found in Fig. 4(b) that the CAS server takes an extra latency in the range 2.98-3.57s to generate an extended certificate for 100 objects. Therefore, the total time to retrieve the permissions and do the file transfer is comparable across the two systems for up to 8 clients. Instead, for 26 concurrent clients, the CAS system only requires 2.4s for the file transfer, while Nephele takes 14.96s to retrieve permissions and transfer the file. However, the CAS server may take minutes to retrieve the permissions for larger numbers of objects or groups, as we already showed in Fig. 4(b) and Fig. 5.

In order to simplify the comparison with CAS, when a transfer request arrives at the Nephele GridFTP server, our current implementation also retrieves the policies that apply to all the objects that are accessible by the hypergroups of the requesting user before matching the file currently requested. However, it is possible to optimize the Nephele database statement so that it selects the permissions that only match the requested file. Then, the curve of Fig. 6 that applies to 1 object (instead of 100) better illustrates the actual authorization cost involved in single file transfers over Nephele. A similar modification could be applied over the CAS server to generate extended certificates with permissions applicable to one file. Nonetheless, this approach is not that beneficial for the CAS-based certificates, because it incurs more frequent requests to the CAS server as the accessed files change. Another possible improvement that Nephele introduces is the placement of the authorization databases locally at each file service, which distributes the authorization load across the fileservers. In fact, ACLs are typically maintained within the fileserver itself, which further lowers the actual authorization cost in more mature systems [30, 47, 56].

7.4 Discussion

In order to get insight about the implications of a real deployment over the Internet, we also emulated the impact of higher round-trip times across the network connections. For that purpose, we used the *netem* implementation of the NISTnet package as made available in the Linux kernel distribution. We applied a 50ms round-trip time from multiple clients to a CAS server handling requests to issue extended certificates. In comparison to our previous experiments, we noticed an increase of the request latency in the range 10-21% as a result of the longer round-trip time. The corresponding increase in the case of Nephele credentials is about 14%. Consequently, we anticipate additional benefits in low service delays if we apply the Nephele model that keeps the authentication service locally within each security domain and avoids long round-trip times.

Overall, embedding the access policies into the extended certificate can lead to high authorization delays at the central CAS server. It is possible to only include a subset of the applicable policies into the certificate, but then the user has to request different certificates more frequently depending on the particular files that he accesses each time. The Nephele approach keeps constant the cost of credential generation regardless of the number of accessible objects. Nonetheless, Nephele incurs the extra delay to retrieve the policies online at the GridFTP server while serving the file transfer request. However, the database access of the Nephele system is amenable to several optimizations, which substantially reduce the incurred latencies in typical production systems.

8 Related Work

Grid Computing. Foster et al. proposed a general architecture to address security issues in distributed computational environments [17]. The users access remote resources through temporary certificates, while domains map remote users to local accounts. The maintenance of costly mappings across numerous entities poses scalability concerns. Instead, the Community Authorization Service (CAS) is a trusted third party that centrally manages the users, resources and policy statements of the collaboration [48]. Role support in the CAS system was described by Cannon et al. [11] and Pereira et al. [49]. Nevertheless, the central server of CAS can become a bottleneck in large installations [27].

The Virtual Organization Membership Service (VOMS) is an authorization system for grid environments that distributes access policies across the resource domains, but it still uses a central server to maintain user identities, group memberships and role assignments [2]. A national federated datastore is currently deployed with VOMS-based certificate management [22]. The ability to map grid credentials to Unix credentials is available to VOMS through the Local Credential Mapping Service [33]. As suggested by previous research for a global file system [30], the Nephele architecture avoids the cost of this mapping by allowing a remote entity (e.g. hypergroup) to directly appear in the access control list of a local file.

Shibboleth is a single sign-on mechanism for web-based applications. It provides secure attribute transport that allows a user to access protected resources at remote domains via a web browser [12]. The GridCertLib library leverages a Shibboleth infrastructure to generate short-lived certificates for seamless access to grid resources [42]. Short-lived certificates are also considered attractive for cross-domain computation scheduling in hybrid environments that run grid middleware on-demand over virtual machines [8]. A secure virtual enclave identifies a distributed collection of related resources along with the principals that are authorized to access them [57]. A virtual enclave uses principal recognition rules and local policies to authorize remote accesses. Rule updates among the enclaves raises issues of maintenance cost and policy consistency.

Role-based Access Control. The role is a semantic construct that allows an organization to assign access permissions to users based on job functions [55]. Role-based access control (RBAC) facilitates user-permission assignments, because roles are relatively persistent with respect to user turnover and task reallocation. Role hierarchy is a structure of roles that can reflect the line of authority in an organization [54]. Role-based administration introduces a special hierarchy of administrative roles to consistently decentralize the role management [53]. Alternatively, a role hierarchy can be extended and flexibly managed with administrative roles through the concept of administrative scope [15]. In order to facilitate the assignment of permissions to roles, an organizational structure can be directly used to define pools of users and permissions independently of role hierarchies [45]. Roles differ from user groups, because a group only consists of users, while a role is a collection of both users and permissions. Moreover, role-based access control has limited support in the ordinary operating systems and fileservers that we target [26].

Distributed Filesystems. The Self-certifying File System (SFS) addresses the problem of user authentication in a global filesystem [30, 37]. The authors disregard certificate authentication infrastructures and chain discovery algorithms because they are hard to deploy. Instead, they allow users to create personal groups that contain remote users and groups. Authorization at the fileserver relies on access control lists (ACLs) of local and remote users that are updated automatically over the network. The caching of remote group membership lists has been additionally evaluated elsewhere [24]. A comprehensive survey of decentralized access control in distributed filesystems has been published recently by Miltchev et al. [41]. Previously, Crisis was introduced as a wide-area authentication and access control system [7]. Certificates signed by a certification authority (CA) are trusted by both endpoints of a communication channel [38]. Hierarchically organized CAs allow a principal to believe a certificate endorsed by a foreign domain, if a path of trust exists between the local and remote domain [9]. Crisis uses ACLs and relies on certificates to create groups and specify group membership [43].

The Andrew distributed file system originally defined the access permissions within a protection domain in terms of users and user groups [56]. In a later version of Andrew, authentication was handled by the Kerberos system, which provided inflexible support for cross-domain collaboration [44]. Leung et al. introduce the Maat system where they describe hash-based capabilities to authorize any number of clients for any number of files [34]. Secure multi-tenant storage for filesystem cloud services has been experimentally investigated but without consideration of the federation dimension across different domains that we examine [32]. A recent study relies on novel key distribution techniques to manage the user groups of ACLs over completely insecure fileservers in cloud environments [50]. Instead, we assume that individual fileservers are relatively secure and rely on hypergroups to distribute group management among different domains.

Trust Management. Unlike certificate-based security systems that bind keys to identities, a trust management system uses credentials that bind public keys to authorized actions and make unnecessary the mappings between identity and authority [10]. For instance, DisCFS applies trust management credentials to identify users, files and rights in remote file accesses [40]. Also, trust management provides a general framework to enforce uniform security policies over large-scale distributed environments [25]. Clarke et al. introduce algorithms for chain discovery to prove that the public key of a client belongs to the groups of an ACL or is delegated authority from another key [14]. Li and Mitchell combine trust management with roles and provide an algorithm for goal-directed chain discovery over distributed credential storage [35]. Freudenthal et al. propose resource authorization in coalition environments through cross-domain role delegation [18]. In large-scale networks, Keromytis and Smith suggest to distribute policies through public-key credentials that enclose authorization attributes, and handle administration complexity through multi-layer hierarchical combination of policies [31]. Nephele also uses credentials to securely bind together sets of public keys, but it stores access permissions locally at each fileserver.

Distributed Systems. Dekker et al. introduce formal requirements for role-based administration in distributed systems [16]. They also propose a procedure to deploy policy changes across security domains with different objects. Li et al. suggest that system-level administrators assign users to groups, while group-level administrators assign roles to the users of each group [36]. The authors call *virtual group* a collection of roles contributed by collaborating domains, and propose algorithms to automatically handle role conflicts. Jøsang et al. provide a comprehensive survey of digital identity management for online services [28]. Tolone et al. compare existing models of access control for collaborative environments through several assessment criteria [61].

SecPAL is a declarative authorization language that can express policy idioms, including authority delegation and domain-specific constraints, to automate authorization decisions over distributed systems [6]. The PERMIS system uses X.509 attribute certificates to centrally manage user roles and authorization policies [13]. The Akenti system uses a public-key infrastructure to issue certificates that identify users, specify policies and assign access rights [60]. SESAME allow users to access remote domains through certificates that contain role-based privileges [3,29]. The xDAuth system relies on a third-party delegation service to authorize requests based on the policy of the resource domain and the authentication of the user domain [1]. The OASIS architecture enables remote access through role-membership certificates that users conditionally accumulate from different services [5, 23]. For policy flexibility, OASIS replaces ACLs with statements written in a role-definition language. Overall, the above schemes use permission certificates, roles or central control to serve the general needs of open distributed systems, while Nephele introduces the hypergroups as a decentralized mechanism for the scalable maintenance of access control lists in distributed file services (Section 2).

9 Conclusions and Future Work

Distributed access control over federated file services can be hurdled through heavyweight updates among independent security domains. We introduce hypergroups as two-layer constructs of user groups that span multiple domains and only involve domain membership updates between different domains. Then, we propose the Nephele architecture to keep the user authentication and action authorization services local at each domain. We built a prototype Nephele system and comprehensively experimented with authentication and authorization requests for GridFTP transfers. In comparison to centralized systems, Nephele increases moderately the processing latency of the GridFTP requests, but makes the cost to issue credentials lower and independent of the number of accessible objects. Reduced cross-domain update traffic and distribution of the authentication and authorization across the domains improve the scalability of the system. We also confirm analytically the comparative scalability benefits of Nephele with respect to several representative architectures. Based on the positive outcome of the present study, in our future work we aim to explore the scalability properties of Nephele across a broader range of storage and communication technologies, including wireless devices and cloud environments, where the low update traffic of our approach makes it a potentially attractive access-control method.

Acknowledgements We are thankful to the anonymous reviewers for their insightful comments that helped us improve the manuscript. Nikolaos Mpountouropoulos provided early measurements on a preliminary prototype of Nephele. In part this research has been supported by project Interstore of the Interreg IIIA Greece-Italy EU Community Initiative Programme under contract No. I2101005 co-funded by the European Union and the Greek State.

References

1. Alam, M., Zhang, X., Khan, K.H., Ali, G.: xDAuth: a scalable and lightweight framework for cross domain access control and delegation. In: ACM Symposium on Access Control Models and Technologies. Innsbruck, Austria (2011)
2. Alfieri, R., Cecchini, R., Ciaschini, V., dell' Agnello, L., Frohner, A., Lorentey, K., Spataro, F.: From gridmap-file to VOMS: managing authorization in a grid environment. Future Generation Computer Systems (Elsevier) **21**, 549–558 (2005)
3. Ashley, P.: Authorization for a large heterogeneous multi-domain system. In: Australian Unix and Open Systems Group National Conference, pp. 159–169. Brisbane, Australia (1997)
4. Avetisyan, A.I., Campbell, R., Gupta, I., Heath, M.T., Ko, S.Y., Ganger, G.R., Kozuch, M.A., O'Hallaron, D., Kunze, M., Kwan, T.T., Lai, K., Lyons, M., Milojicic, D.S., Lee, H.Y., Soh, Y.C., Ming, N.K., Luke, J.Y., Namgoong, H.: Open Cirrus: A global cloud computing testbed. Computer **43**, 35–43 (2010)

5. Bacon, J., Moody, K., Yao, W.: A model of oasis role-based access control and its support for active security. ACM Transactions on Information and System Security **5**(4), 492–540 (2002)
6. Becker, M.Y., Fournet, C., Gordon, A.D.: Design and semantics of a decentralized authorization language. Journal of Computer Security (IOS Press) **18**(4), 619–665 (2010)
7. Belani, E., Vahdat, A., Anderson, T., Dahlin, M.: The CRISIS wide area security architecture. In: USENIX Security Symposium, pp. 15–30. San Antonio, TX (1998)
8. Birkenheuer, G., Brinkmann, A., Högqvist, M., Papaspyrou, A., Schott, B., Sommerfeld, D., Ziegler, W.: Infrastructure federation through virtualized delegation of resources and services. Journal of Grid Computing (Springer) **9**(3), 355–377 (2011)
9. Birrell, A.D., Lampson, B.W., Needham, R.M., Schroeder, M.D.: A global authentication service without global trust. In: IEEE Symposium on Security and Privacy, pp. 223–230. Oakland, CA (1986)
10. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: IEEE Symposium on Security and Privacy, pp. 164–173. Oakland, CA (1996)
11. Cannon, S., Chan, S., Olson, D., Tull, C., Welch, V., Pearlman, L.: Using CAS to manage role-based VO sub-groups. In: Intl Conference on Computing in High Energy and Nuclear Physics. La Jolla, CA (2003)
12. Cantor, S.: Shibboleth architecture: Protocols and profiles (2005). URL http://shibboleth.internet2.edu/. Internet2/MACE
13. Chadwick, D.W., Otenko, A.: The PERMIS X.509 role based privilege management infrastructure. In: ACM Symposium on Access control Models and Technologies, pp. 135–140. Monterey, CA (2002)
14. Clarke, D., Elien, J.E., Ellison, C., Fredette, M., Morcos, A., Rivest, R.L.: Certificate chain discovery in SPKI/SDSI. Journal of Computer Security (IOS Press) **9**, 285–322 (2001)
15. Crampton, J., Loizou, G.: Administrative scope: A foundation for role-based administrative models. ACM Transactions on Information and System Security **6**(2), 201–231 (2003)
16. Dekker, M., Crampton, J., Etalle, S.: RBAC administration in distributed systems. In: ACM Symposium on Access Control Models and Technologies, pp. 93–101. Estes Park, CO (2008)
17. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: ACM Conference on Computer and Communication Security, pp. 83–92. San Francisco, CA (1998)
18. Freudenthal, E., Pesin, T., Port, L., Keenan, E., Karamcheti, V.: dRBAC: distributed role-based access control for dynamic coalition environments. In: IEEE Intl Conference on Distributed Computing Systems, pp. 411–420. Vienna, Austria (2002)
19. Galvin, J.M.: Public key distribution with secure DNS. In: USENIX Security Symposium, pp. 16–25. San Jose, CA (1996)
20. The Globus Alliance: GT 4.0.7 CAS Developer's Guide. March 2008
21. The Globus Alliance: GT 4.0.7 GridFTP Developer's Guide. March 2008
22. Goghlan, B., Walsh, J., Childs, S., Quigley, G., O'Callaghan, D., Pierantoni, G., Ryan, J., Simon, N., Rochford, K.: The back-end of a two-layer model for a federated national datastore for academic research VOs that integrates EGEE data management. Journal of Grid Computing **8**(2), 341–364 (2010)
23. Hayton, R.J., Bacon, J.M., Moody, K.: Access control in an open distributed environment. In: IEEE Symposium on Privacy and Security, pp. 3–14. Oakland, CA (1998)
24. Hemmes, J., Thain, D.: Cacheable decentralized groups for grid resource access control. In: IEEE/ACM Intl Conference on Grid Computing, pp. 192–199. Las Vegas, NV (2006)
25. Ioannidis, S., Bellovin, S.M., Ioannidis, J., Keromytis, A.D., Smith, J.M.: Design and implementation of virtual private services. In: IEEE Intl Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 269–274. Linz, Austria (2003)
26. Jaeger, T.: Operating System Security. Synthesis Lectures on Information Security, Privacy and Trust. Morgan & Claypool (2008)
27. Jie, W., Arshad, J., Townend, P., Lei, Z.: A review of grid authentication and authorization technologies and support for federated access control. ACM Computing Surveys **43**(2), 12:1–12:26 (2011)
28. Jøsan, A., Zomai, M.A., Suriadi, S.: Usability and privacy in identity management architectures. In: Australasian Information Security Workshop: Privacy Enhancing Technologies, pp. 143–152. Ballarat, Australia (2007)
29. Kaijser, P., Parker, T., Pinkas, D.: Sesame: The solution to security for open distributed systems. Computer Communications **17**(7), 501–518 (1994)

30. Kaminsky, M., Savvides, G., Mazières, D., Kaashoek, M.F.: Decentralized user autheni-cation in a global file system. In: ACM Symposium on Operating Systems Principles, pp. 60–73. Bolton Landing, NY (2003)
31. Keromytis, A.D., Smith, J.M.: Requirements for scalabale access control and security management architectures. ACM Transactions on Internet Technologies **7**(2), 1–22 (2007)
32. Kurmus, A., Gupta, M., Pletka, R., Cachin, C., Haas, R.: A comparison of secure multi-tenancy architectures for filesystem storage clouds. In: ACM/IFIP/USENIX International Middleware Conference. Lisboa, Portugal (2011)
33. Lcmaps. http://wiki.nikhef.nl/grid/LCMAPS, April 2012
34. Leung, A.W., Miller, E.L., Jones, S.: Scalable security for petascale parallel file systems. In: ACM/IEEE Conference on Supercomputing (SC), pp. 1–12. Reno, NV (2007)
35. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust-management framework. In: IEEE Symposium on Security and Privacy, pp. 114–130. Berkeley, CA (2002)
36. Li, Q., Zhang, X., Xu, M., Wu, J.: Towards secure dynamic collaborations with group-based RBAC model. Computers & Security (Springer) **28**, 260–275 (2009)
37. Mazières, D., Kaminsky, M., Kaashoek, M.F., Witchel, E.: Separating key management from file system security. In: ACM Symposium on Operating Systems Principles, pp. 124–139. Kiawah Island, SC (1999)
38. ITU-T X.509 Recommendation: Information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks. International Telecommu-nication Union, Geneva, Switzerland (2005)
39. ITU-T X.903 Recommendation: Information processing - open distributed processing - ref-erence model: Architecture. International Telecommunication Union, Geneva, Switzerland (1996)
40. Miltchev, S., Prevelakis, V., Ioannidis, S., Ioannidis, J., Keromytis, A.D., Smith, J.M.: Secure and flexible global file sharing. In: USENIX Annual Technical Conference, Freenix Track, pp. 168–178. San Antonio, TX (2003)
41. Miltchev, S., Smith, J.M., Prevelakis, V., Keromytis, A., Ioannidis, S.: Decentralized access control in distributed file sysems. ACM Computing Surveys **40**(3), 10:1–10:30 (2008)
42. Murri, R., Kunszt, P.Z., Maffioletti, S., Tschopp, V.: GridCertLib: a single sign-on solution for grid web applications and portals. Journal of Grid Computing (Springer) **9**(4), 441–453 (2011)
43. Neuman, B.C.: Proxy-based authorization and accounting for distributed systems. In: IEEE Intl Conf on Distributed Computing Systems, pp. 283–291. Pittsburgh, PA (1993)
44. Neuman, B.C., Ts'o, T.: Kerberos: An authentication service for computer networks. IEEE Communications Magazine **32**(9), 33–38 (1994)
45. Oh, S., Sandhu, R., Zhang, X.: An effective role administration model using organization structure. ACM Transactions on Information and System Security **9**(2), 113–137 (2006)
46. Osborn, S., Guo, Y.: Modeling users in role-based access control. In: ACM Workshop on Role-based Access Control, pp. 31–37. Berlin, Germany (2000)
47. Pawlowski, B., Noveck, D., Robinson, D., Thurlow, R.: The NFS version 4 protocol. In: SANE Intl System Administration and Networking Conference. Maastricht, Netherlands (2000)
48. Pearlman, L., Welch, V., Foster, I., Kesselman, C.: A community authorization service for group collaboration. In: Intl Workshop on Policies for Distributed Systems and Networks, pp. 50–59. Monterey, CA (2002)
49. Pereira, A.L., Muppavarapu, V., Chung, S.M.: Managing role-based access control policies for grid databases in OGSA-DAI using CAS. Journal of Grid Computing (Springer) **5**(1), 65–81 (2007)
50. Popa, R.A., Lorch, J.R., Molnar, D., Wang, H.J., Zhuang, L.: Enabling security in cloud storage SLAs with CloudProof. In: USENIX Annual Technical Conference, pp. 355–368. Portland, OR (2011)
51. Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., Scavo, T.: Security Assertion Markup Language (SAML) V2.0 Technical Overview. Organization for the Advancement of Structured Information Standards (OASIS) (2007)
52. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Pro-ceedings of the IEEE **63**(9), 1278–1308 (1975)
53. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based admin-istration of roles. ACM Transactions on Information and System Security **2**(1), 105–135 (1999)

54. Sandhu, R., Ferraiolo, D., Kuhn, R.: The NIST model for role-based access control: towards a unified standard. In: ACM Workshop on Role-based Access Control, pp. 47–63. Berlin, Germany (2000)
55. Sandhu, R.S., Coyne, E.J.: Role-based access control models. Computer (IEEE) **29**(2), 38–47 (1996)
56. Satyanarayanan, M.: Integrating security in a large distributed system. ACM Transactions on Computer Systems **7**(3), 247–280 (1989)
57. Shands, D., Yee, R., Jacobs, J., Sebes, E.J.: Secure Virtual Enclaves: supporting coalition use of distributed application technologies. ACM Transactions on Information and System Security **4**, 103–133 (2000)
58. Stribling, J., Sovran, Y., Zhang, I., Pretzer, X., Li, J., Kaashoek, M.F., Morris, R.: Flexible, wide-area storage for distributed systems with wheelfs. In: USENIX Symposium on Networked Systems Design and Implementation, pp. 43–58. Boston, MA (2009)
59. Taiani, F., Hiltunen, M., Schlichting, R.: The impact of web service integration on grid performance. In: IEEE Intl Symp on High Performance Distributed Computing, pp. 14–23. Research Triangle Park, NC, USA (2005)
60. Thompson, M.R., Essiari, A., Mudumbai, S.: Certificate-based authorization policy in a PKI environment. ACM Transactions on Information and System Security **6**(4), 566–588 (2003)
61. Tolone, W., Ahn, G.J., Pai, T.: Access control in collaborative systems. ACM Computing Surveys **37**(1), 29–41 (2005)
62. Trostle, J.T., Neuman, B.C.: A flexible distributed authorization protocol. In: Symposium on Network and Distributed System Security (Internet Society), pp. 43–52. Internet Society, San Diego, CA (1996)