P-Hera: Scalable fine-grained access control for P2P infrastructures

Bruno Crispo, Swaminathan Sivasubramanian
Department of Computer Science
Vrije Universiteit, Amsterdam
{crispo,swami}@cs.vu.nl

Pietro Mazzoleni
Department of Computer Science
University of Milan,Italy
mazzolen@dico.unimi.it

Elisa Bertino
Department of Computer Science
Purdue University and Cerias, USA
bertino@cerias.purdue.edu

Abstract

In this paper, we present P-Hera, a peer-to-peer (P2P) infrastructure for scalable and secure content hosting. P-Hera allows the users and content owners to dynamically establish trust using fine-grained access control. In P-Hera, resource owners can specify fine-grained restrictions on who can access their resources and which user can access which part of data. We differentiate our work with traditional works of fine-grained access control on Web services, as our system in addition to handling access constrains of the service provider (which is the case in Web services), it also handles security constrains regarding actions performed on data: replication and modification. We believe this is of immense significance for wide-range of applications such as data Grids, Information Grids and Web Content Delivery Networks. In addition to presenting the overall system architecture, we also study the problem of evaluating these fine-grained access policies in depth and propose a novel means of organizing these policies that can result in faster evaluation. We demonstrate the effectiveness of our approach using prototype implementation.

1. Introduction

1.1. Background

During the last years, Grid systems have gained immense interest in both research and industry community. The reason behind the interest is that Grid enables resource sharing and collaboration across many resources in the Internet. This allows users to focus on the application without having to be worried about on which machine the application code will run, provided that their trust requirements are met.

While the Grid started as an infrastructure that allows seamless access to remote resources for executing computationally intensive jobs, it is now evolving to new Grid applications such as data Grid and information Grid. Compared to traditional computing infrastructures, data Grid aims at finding a reliable set of nodes that can host and manage a particular piece of data. Possible applications of data Grid include replication of documents, as in Content Delivery Networks [16], or storing scientific data to enhance reliability and to provide convenient access to remote computations. Similarly, information Grid aims at enabling seamless access to trusted information and is being considered as the next generation World-Wide Web.

To implement data and informational Grid infrastructure, researchers have recently advocated the possibility of using peer-to-peer (P2P) technologies [8] and some projects started to appear [10, 11]. In fact, P2P technologies (e.g., Chord [17], Pastry [15], Gnutella and Kazaa [12]) provide scalable decentralized infrastructures which well address issues of scalability and reliability of the processes of storing and locating information.

However, an effective deployment and large scale use of P2P technologies in applications dealing with content distribution require a proper security framework and access control to be in place. Such a requirement is crucial given the increasing value that digital contents today has. Usually, P2P infrastructures assume that all participants of a P2P network are equally 'trustable' and thereby it remains unsuitable for applications such as wide-area storage for Grid-like systems or Content Delivery Networks (CDNs) [16] due to security concerns. To promote the usage of those systems in business domain, we therefore believe each subject should be able to express its own authorization constrains based on the role played in the network and accept only the other parties' constrains only if it feels appropriate. Possible roles a subject can play in the network are *Resource Owner* (RO), which makes available its data storage to collect user information, *Data Owner* (DO), which has the ownership of a data made available into the network, and *Content client* (CC), which is the subjects requesting access to data.

In this paper, we propose the design of P-Hera, a P2P infrastructure that allows ROs, DOs, and CCs to dynamically establish trust using fine-grained access control. In P-Hera, ROs can specify fine-grained restrictions on which DOs can replicate data into their resources and which CCs can access which part of data. For example, resource owners from Vrije Universiteit can stipulate that only users from IBM can access their resources. Similarly, DOs can specify access constrains on where (which ROs) their information/data can be replicated and which CCs are allowed to read their information.

A naive implementation of FGAC is to let each node evaluating each and every user request to see whether it meets its access control constrains. However, this might result in erroneous scheduling of requests as users are not aware of the policy constrains of the nodes they manage. This can lead to increased bandwidth utilization, as eventually the request needs to be forwarded to each and every node, and turn-around time for each request, as contacted nodes may reject a request.

For the same reason, FGAC for P2P systems warrants a smart organization of resources' FGAC policies so that the process of policy evaluation of resources do not result in a scalability bottleneck.

P-Hera efficiently organizes FGAC policies so that requests for information/data/resources are evaluated much faster. As the results of our preliminary performance evaluation experiments show, our policy organization results in an order of magnitude speed up in the policy evaluation time and avoid situations in which user requests are sent to a node that cannot execute the operation because of the lack of authorization.

The contributions of this paper are thus threefold. First, we identity the need of supporting FGAC policies in P2P infrastructures and present the design and implementation of a Grid infrastructure to specify and enforce such policies. Second, we propose a scalable evaluation strategy for these policies which is integrated with the resource search activity. Third, we present and compare different algorithms for the evaluation of fine-grained policies, and determine which is the most efficient policy evaluation algorithm.

The rest of the paper is organized as follows. Section 2 describes the system architecture and briefly introduces the methodology we propose to support FGAC in the Grid. Section 3 discusses the schema according to which policies are organized. Section 4 presents the implementation of the policy evaluation strategies and reports performance evaluation results. Section 5 reviews the state of the art while Section 6 concludes the paper and outlines future work.

2. System Architecture

2.1. System Model

The development of a subsystem supporting fine access control policies for a P2P system requires the identification of a reference architecture, among the various proposed architectures, and of a set of system primitives, dealing with data management. In the remainder of this subsection we thus define the system model we assume in our work.

2.1.1. System infrastructure In this paper, we restrict our design to unstructured P2P networks for the afore stated advantages. Among unstructured P2P networks, there are traditional complete unstructured networks such as Gnutella where there is no organization and any search is executed by broadcasting the query to all nodes in the network. This can result in a huge waste of network bandwidth. A variant of this model is to introduce some hierarchy in which a special set of nodes is dynamically elected to manage the other nodes in the network. These nodes, usually called *supernodes*, store the directory information of the contents of the nodes they manage. In such systems, a query is then answered only by a subset of nodes and the amount of messages is reduced by two to three orders of magnitude. In this paper, we assume such a hierarchical unstructured P2P infrastructure and we investigate how to extend it with fine-grained access control.

2.1.2. System primitives In general, a P2P infrastructure does the following basic operations:

- *search(data)*: This operation supports searches for a specific data item in the system. Such a query is sent from a user, possibly through a web interface, to its representative supernode. This supernode queries other supernodes and each supernode returns the list of nodes under its management that have the required data.
- *get(data)*: This operation concerns retrieving the data from one of the nodes that has it (result of the previous primitive). In a traditional P2P network, this just boils down to establishing a direct network connection to the node.
- *place(data)*: This operation supports placing a replica of a data item in a P2P network. Replication is usually adopted to improve reliability, in the case the data owner is not available, and performance in that data is located closer to users as in CDNs. Data placement requires selecting a node that satisfies the resource constrains, such as disk space, availability, and bandwidth, of the user requesting a replica.

In this paper, we present the design of a system that integrates FGAC in the context of these three primitives. Whereas several research efforts have been devoted to the performance optimization for each of these primitives, very little work has been carried out dealing with the extension of these primitives with FGAC.

2.2. Fine-grained Authorization policy

Authorization policies, or *policies*, are logical expressions specifying the constrains used to restrict access to a given resource. By abstracting from the language used, a policy can be defined as the combination of one or multiple *policy rules*. Each policy rule pol_rul_i is defined as a pair $\langle pol\_exp_i, Effect_i \rangle$ where pol_exp_i is a logical expression built using a *policy attribute* (e.g., User.Name, User.Affiliation, Resource.Amount, Time), and $Effect$ represents the action (Permit or Deny) to be executed when the policy expression is verified. Some examples of policy rules are $\langle Time, (TimeInRange, 8am \ldots 5pm), Permit \rangle$, $\langle (Affiliation, (StringEqual, Unimi), Permit \rangle$ which restrict access only to users whose Affiliation is Unimi and requests being entered from 8am to 5pm. Multiple rules composing a policy are combined via combination algorithms such as Deny Override (i.e., a request is denied if even one rule is denied), Permit Override, (i.e., a request is authorized if even one rule is permitted), First Rule Applicable, Only One Applicable, and so on. Recently, several alternatives have been proposed as standard language to define policies. XML-based languages such as XACML, SAML, WS-Policy and Author-X [21] [5] [1] [7], are all rich in semantics and very flexible, thus allowing resource owners to specify articulated access control policies. Although our solution can be applied to any of the above languages, we focus our attention on XACML. XACML - Extensible Access Control Markup Language, is the standard OASIS to specify authorization policies; it supports all the above combination algorithms and each policy rule can be issued to restrict a specific operation (e.g., read, write) on a single resource. Figure 1 shows an example of policy specified by using such formalism.

FGAC is therefore intended not only as a detailed list of constrains to restrict access to a resource, but also as the possibility to specify different authorizations for each possible action and each possible resource owned by a user. It is important to notice that in P2P systems, both data owner (DO) and resource owner (RO) can specify access control policies. The former can place constrains about where its data can be placed and which users can access them, whereas the latter can place constrains on which users can replicate data in its resources and who can access data

```
<Policy PolicyId=" Example_of_Hosting _Policy"
   RuleCombiningAlgId="deny -overrides">
<Description> XACML Policy composed by 2 policy rules
 </Description>
  <Target>
   <Subjects>    <AnySubject/>    </Subjects>
   <Resources> <AnyResource/> </Resources>
   <Actions>      <Write>      </Actions>
  </Target>
  <Rule RuleId="Condition_On_Affiliation" Effect="Permit">
   <Condition FunctionId="  function:string-equal">
    <Apply FunctionId="function:string -one-and-only">
     <SubjectAttributeD esignator   AttributeId="Affiliation"
         DataType=" #string"/>
    </Apply>
    <AttributeValue DataType="#string">Unimi   </AttributeValue>
   </Condition>
  </Rule>
  <Rule RuleId="Condition_On_Time" Effect="Deny">
   <Condition FunctionId="  function#time-in-range">
    <Apply FunctionId="  function:time-one-and-only">
     <EnvironmentAttributeDesignator AttributeId="current  -time"
         DataType="#time"/>
    </Apply>
    <AttributeValue DataType="  #time">08:00:00</AttributeValue>
    <AttributeValue DataType=" #time">20:00:00</AttributeValue>
   </Condition>
  </Rule>
</Policy>
```

Figure 1. XACML authorization Policy

hosted into its resources. In addition to these constrains, users requiring access to contents, referred to as data readers, can also specify restrictions to avoid receiving information from nodes they do not trust. In this context, we distinguish three types of policies: (i) *Placement policy*: specified by the DO, it concerns the nodes where data can/cannot be replicated, (ii) *Hosting policy*: specified by the RO, it concerns which DO can replicate its information in its resource, and (iii) *Access control*: specified by both RO (and user) regarding who can access information replicated in its resource (or for the user, which resource it trusts in obtaining information). Note that these three different types of policy and their corresponding players differentiate our system from the traditional client-server model of access control in Web services. With respect to the primitives introduced earlier, hosting and access policies are relevant for the get(data) primitive, while placements constrains are relevant to the place(data) primitive.

2.3. System Architecture

The proposed system architecture is given in Figure 2. As shown by the figure, we consider an hierarchical unstructured P2P system the main components of which are supernodes, nodes and users.

In our system, supernodes, in addition to maintaining indices of data contained in the nodes, also maintain the access control policies of the nodes under their management. Supernodes have the following components: *query evaluator*, *content manager* and *policy manager*. The *query evalu-*

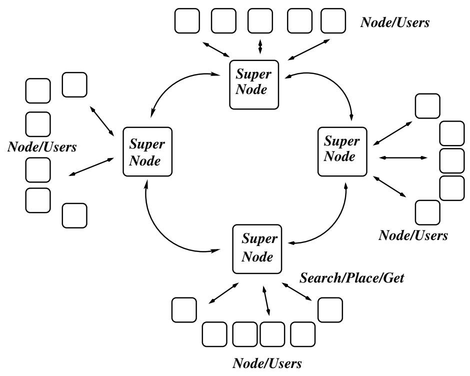

Figure 2. System architecture

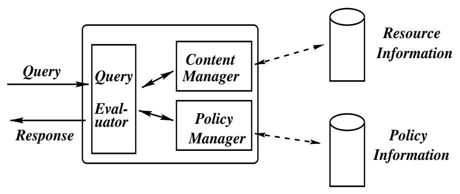

Figure 3. SuperNode Architecture

ator has the same functionalities as in the conventional supernodes. Its responsibility is to receive queries from other supernodes and to perform appropriate actions. The *content manager* component is in charge of finding the set of ROs, among the ones administered by the supernode, that have the data requested by the query evaluator. Finally, the *policy manager* determines the set of ROs, among the ones selected by the content manager, that authorize the request by evaluating the access policies of the DO, ROs and CCs.

We now briefly look at how the system performs the above described basic operations.

- *Search*: For searching a data item, a user submits his profile information concerning his identify (in addition to information regarding the requested data) to the supernode of his node. The supernode forwards the query along with the user profile to other supernodes. These supernodes in turn match the user profile with the access control policies of the nodes under their management and return the nodes which not only have the required data item but also the ones that authorizes the user's request.
- *Get:* When a user actually contacts a node for getting a data, the user submits his identity and the node can decide to check the user privileges again or not, depending on whether it trusts the decisions of the supernode, before serving the content.
- *Placement:* Like the search operation, each user forwards the placement request and his placement constrains to the supernode of his node, which in turn forwards it to other supernodes. Each of these supernodes checks the hosting policies of its nodes in addition to the resource constrains and returns the list of nodes that authorize the node's placement request.

3. Policy Management

Because of the large number of nodes and users and the different types of policy we support in our system, policy management is a crucial issue. Two are the most relevant issues concerning the management of FGAC policies: *policy storage* and *policy organization*. The first issue concerns where the policies of the individual nodes and users must be stored. The second issue deals with how these policies must be organized so that the evaluation process of multiple policies can be efficiently performed. We discuss our solutions in the following sections.

3.1. Policy Storage

A naive approach to policy storage is to store the policies in the respective nodes. Thereby, each RO stores its own placement and hosting constrains. Under such an approach the three basic data management primitives (i.e., search, get, placement) can be scheduled without taking into account security constrains. This can result in erroneous scheduling of requests by the supernodes as they are not aware of the policy constrains of the nodes they manage. This can lead to increased bandwidth utilization, as eventually the request needs to be forwarded to each and every node, and turnaround time for each request, as nodes that are selected by a supernode may not authorize the request. Hence, such an approach cannot be adopted. We also rule out the opposite approach which consist of storing the policies of all nodes and users in each node for reasons of poor scalability.

In order to develop approaches overcoming the above drawbacks we choice to extend the request schedulers, residing in supernodes, so to be aware of the access control policies of individual nodes. As discussed into the previous sections, each supernode stores the access control and hosting policies of the nodes that it manages. Such an organization makes possible for supernodes the selection of nodes that not only satisfy the resource constrains of the request, such as whether a node has the data or not, whether a node has enough disk space or not, but also that authorize the user's placement or search request. This mechanism can also improve the system scalability as each request is scheduled just by the supernodes without contacting the individual nodes, leading to reduced response times and bandwidth utilization.

Storing the policies in supernodes has the underlying limitation that the individual nodes should trust to some extent the supernode. For example, if a supernode is not trustable and alters the policy of a node, then a node can receive a user request for data access or placement even though the authorization constrains are not met. However, this is not a major problem as the node holds the data can still decide to perform its evaluation of the user request to see if the request is really authorized. Therefore, a node upon receiving a request from a user, can check if it is certified by the supernode (assuming supernodes and nodes store public-private key pair). If so, it can accept the request with or without rechecking authorization constrains.

3.2. Policy Organization

As noted earlier, supernodes need to evaluate the policies of each node for each user request. Such a process can become another scalability bottleneck if there are many users requests and the number of nodes and number of policy expressions within a policy is high, which is typically the case of large-scale P2P infrastructures.

This calls for efficient organizations of information concerning nodes policies at supernodes so that the evaluation process can be much faster. In this section, we present different strategies for organizing and grouping policies and discuss their relative merits/demerits.

1. **No grouping:** Under this strategy policies are not organized or grouped. For each request, the policies of each node are evaluated individually.
2. **Group by policies:** Under this strategy nodes with same policies are grouped together. For each request, evaluations are executed only at the group level. The intuition behind this heuristic is that if many nodes have same policies then a redundant evaluation of these nodes is avoided. This approach is used in Gangmatching [18].
3. **Group by policy expressions:** Under this strategy nodes using the same policy expressions in the definition of their policy are grouped together. For example, consider two nodes that use the policy expression affiliation='IBM'; then these two nodes are grouped together. This strategy is motivated by the fact that in fine-grained access control environments, it is more likely to find nodes that use same policy expressions compared to nodes that use same LP (which can be made of multiple policy expressions).

In the next section, we analyze the performance of each organization and evaluate the response time for evaluating a request under each organization.

4. Evaluation of Policy Organization Strategies

4.1. Implementation

A proof of concept for our system has been built at University of Milan. The implementation focuses on the policy storage organization strategies proposed into Section 3.2 because we believe scalability is the most critical issue when supernodes are required to evaluate authorization policies expressed by a high number of peers.

We have implemented all the three strategies. However, while the first two, *no grouping* and *group by policies* are quite straightforward, we are not aware of any distributed policy engine using the third strategy. We call this strategy as *Policy Expression Grouping* (PEGs). PEGs are defined as a pair $\langle att_i, expr_i \rangle$ where att_i is a policy attribute (e.g., Time, User.Affiliation, etc) and $expr_i$ is a possible logical expression defined using att_i. An example of PEG is $\langle Country, StringEqual = Italy \rangle$. PEGs are dynamically generated based on the occurrence of similar policy rules defined by the nodes.

Such organization offers several advantages in terms of scalability of the evaluation process, confirmed by the preliminary performance evaluation results reported below. In brief: (i) For each node, only a few information (i.e., type of relation between PEG and node) need to be collected instead of the complete LP; (ii) PEGs are defined as policies and they can be evaluated using available policy evaluation systems; (iii) by construction, if a request validates a PEG, it validates the policy rules of the nodes belonging to that PEG.

Our solution has been built extending the XACML implementation developed by Sun (http://sunxacml.sourceforge.net/). SUNXACML 1.2 is the open source project providing complete support for all the mandatory features of XACML. The system is implemented using two components.

The first component is the PEP (Policy Enforcement Point) which is in charge of collecting a request and sending it to the second component, a PDP (Policy Decision Point) which examines the request, and evaluates the policies applicable to the request.

4.2. Performance Evaluation

We have analyzed the evaluation time for each of these three organization methods in our prototype. The experiments were conducted on Pentium4 3.2 Ghz, 1 Gb Ram machine. The OS on this machine is Windows 2000 Professional. The runtime JVM for the processes involved was JRE 1.4.2. In our experiments, a PEG was generated for each pair attribute/value specified by a node.

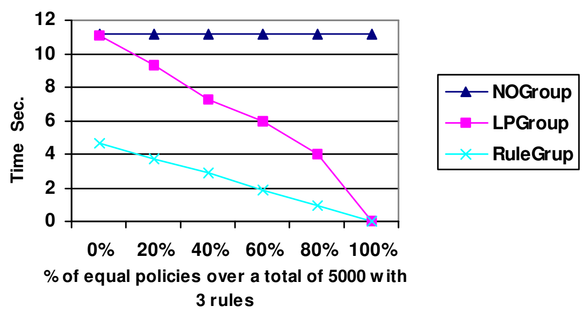

Figure 4. Performance of varying number of equal policies

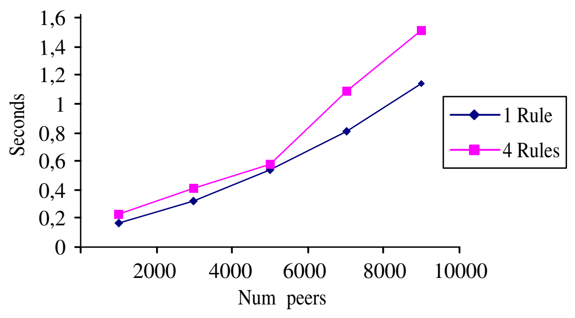

Figure 5. Performance of PEG policy evaluator for different number of policies

In the first experiment, we compared the efficiency of the three strategies in evaluating 5000 policies, a subset of which are equal. Each policy was composed of 3 policy expressions. We varied the attribute values of one of the policy expressions to vary the percentage of equal policies in the set. We evaluated all the three strategies and the results are shown in Figure 4. The results show the strategy of grouping nodes based on similar policy expressions scales better than the others, especially for low similarity ratios. Such behavior is explained by observing that a request can be validated even if it matches one of the PEGs. So, PEG strategy by virtue of grouping nodes by policy expression, identifies precisely which groups need to be checked for validating a given request.

In the second experiment, we studied the effect of the number of rules used in composing a policy on the efficiency of PEG policy evaluation strategy and the results are given in Figure 5. The results show that the impact is minimal, and the PEG strategy scales well in scenarios with simple as well as complex policies.

5. Related Work

The problem of building security in systems that enable resource sharing across large number of nodes has been addressed into many domains such as: *P2P systems*, *Grid systems* and *CDNs*.

P2P systems. Security in P2P systems has been investigated with respect to both the underlying routing protocol and the storage-retrieval process . A considerable amount of research has been carried out for both the areas. Possible routing attacks and available systems preventing them are surveyed in [20]. For security issues concerning the storage-retrieval process, we can distinguish between *user privacy* - addressed for instance by Freenet [4] or Turtle [14] and its aim of creating an "uncensorable and secure global information storage system", and *data security* - which is the problem addressed in this paper. Traditional P2P systems assume a high level of trust between partners, which is inadequate for applications such as Web caches or file sharing across organizations. Subsequently, solutions such as [6, 3] propose finer authorization controls. In [6] the authors apply the concept of Virtual Organization [9]. A finer level of authorization is proposed in [2]. Here, information security is organized at peer level. Each peer can autonomously specify authorization constrains for each information item and the enforcement is performed directly by the node with an ad-hoc implementation of the Akenti authorization system [19] distributed thought the peers. Those solutions are interesting. However, they consider only one of the types of authorization presented into Section 3.2. Specifically, they only support user constrains and do not consider *placement* and *hosting* constrains. We believe this is a serious limitation for the large application of the P2P paradigm.

Grid systems. Grid systems enable resource sharing across multiple administrative domains. However, Grid systems assume that users form virtual communities called virtual organizations [9]. This leads to situations in which all participants are assumed to be in the same trust domain. This actually reduces the social adoption of the system as then participating nodes are forced to trust all Grid users by just joining them. We believe our work is very useful for also data grid systems as it enables users and nodes to establish trust dynamically at a fine-grained level using FGAC policies.

Content Delivery Networks (CDNs). CDNs are specialized distributed systems that have multiple servers placed in prime locations across the Internet hosting multiple copies of a Web site [16]. Each client request is usually served by the closest server. These systems have been traditionally built by a single corporate (e.g., Akamai) that owns all the resources. Recently, there have been proposals to use P2P technologies to perform content distribution in the Internet [10, 11]. However, the impediment to these technologies is the trust relationships among nodes. We believe our mechanism enabling FGAC in combination with secure replication mechanisms such as [13] can enable P2P infrastructures to provide a viable alternative to perform Internet-scale content distribution.

6. Conclusions and Future Work

In this paper, we have described the design of P-Hera, a P2P infrastructure that makes it possible for users and owners of data to dynamically establish trust using fine-grained access control. We have shown how P-Hera enables data owners/resource owners to place/host data while considering security constrains at a very fine-grained level. Furthermore, data can be searched and made available to users by considering the security constrains of CCs, DOs, and ROs. P-Hera has being carefully designed to address scalability and efficiency issues. It can be easily integrated with existing P2P infrastructures and in particular the ones based on supenodes. We have proposed and tested a novel and efficient strategy to organize and evaluate FGAC policies. Based on our preliminary analysis results, we can conclude that existing organization can be improved by adopting our proposed organization. However, we believe more research needs to be carried out dealing with techniques for the scalable evaluation of FGAC policies. We plan to extend our organization to deal with cases in which data, owned by DOs, are stored by ROs that do not grant access to CCs that are customers of these DOs.

Acknowledgements The work of Elisa Bertino is partially supported by the National Science Foundation NSF under the Project “Collaborative Research: A Comprehensive Policy - Driven Framework For Online Privacy Protection: Integrating IT, Human, Legal and Economic Perspectives”, by an IBM Fellowship, and by the sponsors of CERIAS.

References

[1] M. BEA, IBM and SAP. Web servies policy language (ws-policy). 2002.

[2] K. Berket, A. Essiari, and A. Muratas. Pki-based security for peer-to-peer information sharing. *4th International Conference on Peer-to-Peer Computing (P2P 2004) Zurich Switzerland*, 2004.

[3] G. Boella and L. van der Torre. Access control in virtual communities: Prohibition, permission, authorization and delegation of power in the grid. In *Proc. Knowledge Grid and Grid Intelligence workshop at WI/IAT’03 (KGGI’03)*, 2003.

[4] I. Clarke, O. Sandberg, B. Wiley, , and T. Hong. Freenet: A distributed anonymous information,storage and retrieval system. *Int. Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.

[5] O. S. S. T. Committee. Assertions and protocol for the oasis security assertion markup language (saml), Sept 2003.

[6] I. Djordjevic and T. Dimitrakos. Towards dynamic security perimeters for virtual collaborative networks. *Trust Management- Second International Conference (iTrust 2004) Oxford UK*, 2004.

[7] E. F. E. Bertino, S. Castano. On specifying security policies for web documents with an xml-based language. *SACMAT01 ACM Symposium on Access Control Models and Technologies*, 2001.

[8] I. foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proc. of the Second International Workshop on Peer-to-Peer Systems*, 2003.

[9] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[10] J. Hibbard. Forget the music business. in the latest attempt to commercialize peer-to-peer technology, startups want akamai’s market.

[11] D. Kaye. Peer-to-peer content delivery using information additive codecs.

[12] Kazaa. http://www.kazaa.com.

[13] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Secure Data Replication over Untrusted Hosts. In *Proc. 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, pages 121–126. USENIX, May 2003.

[14] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System. In *Proc. 12th Cambridge International Workshop on Security Protocols*. Springer-Verlag, April 2004.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Int. Conf. Distributed Systems Platforms*, pages pp.329–350, Nov. 2001.

[16] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. Replication for web hosting systems. *ACM Computing Surveys*, 36(3):291–334, 2004.

[17] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM ’01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001.

[18] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – a distributed job scheduler. In T. Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[19] M. Thompson, A. Essiari, K.KIeahey, V. Welch, S. Lang, and B. Liu. Fine-grained authorization fo job and resource management using akenti and the globus toolkit. Sept 2003.

[20] D. S. Wallach. A security architecture for computational grids. *International Symposium on Software Security*, 2002.

[21] XAMCL and O. S. S. T. Committee. eXtendible Access Control Markup Language (xacml) committee specification 1.0, Feb 2003.