G-PBox: A Policy Framework for Grid Environments

V. Ciaschini, A. Ferraro, A. Ghiselli, G. Rubini, R. Zappi, INFN CNAF, Bologna Italy
A. Caltroni, INFN PD, Padova, Italy

Abstract

A key feature of Grid environments is sharing computing and storage resources among multiple Virtual Organizations (VOs). This process needs a comprehensive policy framework. Policy management tools have been developed for single farms, but Grids need a more flexible and distributed policy concept.

Usually VOs and local systems share contracts to regulate resource usage, hence complex relationships among these entities implying different types of policies may exist: VO-oriented, local systems-oriented, and a mix of the two. We propose an approach to the representation and management of such policies: the Grid Policy Box (G-PBox) framework. Our approach is based on a set of databases belonging to hierarchically-organised layers distributed on the Grid and VOs structures. Each layer contains at least policies regarding itself. Layers must communicate to each other to accomodate for mixed policies, originating the need for a secure communication service framework – for privacy reasons – with the ability to sort and dispatch various kind of policies to the involved parties.

In this paper we present our first implementation of the G-PBox, and its architecture details, and we discuss the plans for G-PBox-related application and research.

INTRODUCTION

The primary motivation for using the Grid is sharing resources, data and application services. This approach is supported by the concept of Virtual Organization (VO), which is a set of individuals and/or institutions defined by resource sharing rules. However institutions are reluctant to participate in collaborative multi-domain trust environments due to policy enforcement problems. Current software middleware allows organizations to control only resources owned by themselves, typically farms or local systems, and this makes it difficult to enforce policies regarding VOs. In a Grid context it is necessary to manage policies regarding both VOs and resource providers which often have complex relationships based on agreements that regulate resource usage.

In this paper we propose an approach for the representation and management of policies for Grid infrastructures: the Grid Policy Box (G-PBox) framework. The G-PBox design is based on a set of policy repositories hierarchically-distributed to independent administrative-based layers where each layer contains only policies regarding itself. There are at least two layers: the top layer regarding VOs and the bottom one regarding local farms.

There are three different possible kinds of policies: local policies, set by a farm for its own usage, external policies, set by some external entity (e.g. a VO) to regulate farm allocation, and mixed policies, set by the form for its local usage but that also have repercussions at higer layers, e.g. ban-lists. Policies of the latter two types clearly require that there is some form of communication between differen layers.

The approach proposed is focused on security and privacy, so the communication service framework is implemented using the Globus Security Interface (GSI) standard APIs.

The solution proposed does not require hard changes to the current Grid architecture. What is needed is to implement a Policy Enforcement Point (PEP) for each service that wants to be managed by the G-PBox framework.

PROBLEM STATEMENT

The emergence of large-scale grids brings about the need for VOs to tightly regulate both access from the users and the capabilities the users have once logged in on the system. Furthermore, the use of automated systems like resource brokering and job management also needs to be regulated. The obvious solution is to set up a number of access, management and accounting policies.

Existing policy systems have been created for environments where all the resources belong to a single administrative domain, while in a grid infrastructure we have many separate administrative domains. Each one has its own set of policies (local policies) that contain the whole of the agreements with users and VOs. These policies do not need to be known outside the site that originated them, and indeed they may be considered private.

Thus, to present a unified environment to its users, a grid-enabled and grid-aware policy systems requires the presence of external policies, that need to be distributed to a wide subset of all the sites comprising the grid.

REQUIREMENTS

A policy system must be capable of handling at least the following cases.

Handling site policies A site administrator should have absolute control on resources he ownes. He should have a unique interface to enforce policies for his users and he should be able to take actions ranging

from setting a banner list to balancing the use of its resources.

Handling VO policies A VO might want to set policies affecting subsets of its resources and subsets of its users. The policy system should be capable to distribute this kind of policies to local sites, but the sites should have the last word on whether to accept or refuse them. These policies may be highly mutable and need to be distributed to the various farms.

Adding a farm to a grid The policy system should allow adding farms and resources to an already established Grid environment. Furthermore, it should be able to recognize and interface to the added farm's resources.

Adding a new VO to the grid The policy system should be capable of adding new VOs to those already known. However, local sites should be able to choose whether or not to recognize this addition depending on local policies.

Providing granularity The current middleware does not allow any granularity in its policy decisions. It can just accept/deny VO users. The policy system should also be capable of recognizing the internal group organization of a VO like the one defined by an attribute authority, e.g. VOMS. It should be able to force a bias.

Joining different grids It should be possible to join together different working grids, each with their own policy system.

Adapting to new technologies The policy system should be able to manage the new capabilities that will appear when new software is developed or integrated into the grid (Maui over PBS, etc. . .)

PREVIOUS ART

Below you can find a brief overview of some tools currently available and the reason why they do not cover all the features of G-PBox.

PRIMA

PRIMA (the System for Privilege Management and Authorization in Grids developed at Virginia Tech) is a combination of three elements: a privilege-based security model; a dynamic enforcement model based on the combination of the user's privileges with the resource's security policy prior to the assessment of the user's request (dynamic because this combined policy is then valid only for the current request); an enforcement mechanism based on the management of user accounts on-demand via the execution environment (e.g. through file system access control lists and file system quota mechanisms).

PRIMA's focus is on access control i.e. authorization policies.

CAS: Community Authorization Service

CAS is a security service built on the Globus Toolkit *Grid Security Infrastructure* (GSI).

An administrator acquires a GSI credential to represent a community and runs a CAS server with that identity. Resource providers grant privileges to the administrator using local mechanisms (e.g. gridmap files and disk quotas, filesystem permissions, etc.) after authentication and verification that the community's policies are compatible with the resource provider's own policies. Administrators use the CAS to manage the community's trust relationships (e.g. to enroll users and resource providers into the community) and grant fine-grained access control to resources.

A user requesting access to a resource contacts the CAS server which, after authentication, issues a GSI restricted proxy credential with an embedded policy. The user employs the credentials to connect to the resource. The resource then applies its local policy to determine the amount of access granted to the community, and further restricts that access based on the policy in the CAS credentials.

CAS disadvantages are that it completely removes control from site administrators and that it requires a VO to know everything about the layout and internals of its farms.

LCAS: Local Center Authorization Service

LCAS handles authorization requests to a site and the Local Credential Mapping Service (LCMAPS) and provides all local credentials needed for jobs allowed into the site's computing resources. There are three standard authorization modules checking (1st) if a user is allowed to use the resources (via the gridmap file), (2nd) if a user should be banned (3rd) if the grid is open to job submission.

LCAS disadvantages are that it is only a static access control list and that there is no hierarchy because it is deployed on local sites only. VO policies cannot be taken into consideration.

GACL

GACL is a library to manipulate access control lists in a Grid environment.

GACL limitations are the same as LCAS.

ARCHITECTURE

Bird's Eye View

The G-PBox architecture is based upon a composition of modular objects, Policy Boxes (PBox), as shown in picture **??**.

As you can see, there are PBoxes at various levels: VO level, Domain Level, Site level, Farm Level, possible sub-Farm levels, etc. . .

This helps to clearly limit the scope of a particular group of policies. The VO PBox is the authoritative source for VO-wide policies, the Grid PBox is the authoritative source

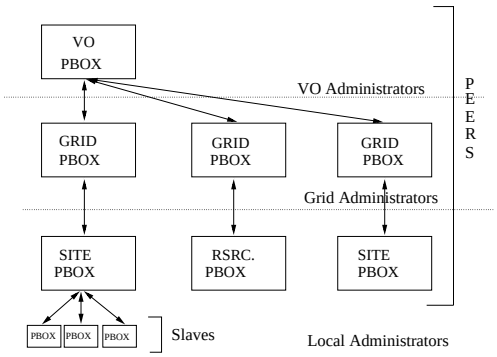

Figure 1: Architecture of G-PBox

for Grid-wide policies, while the Farm PBox is the authoritative source for policies specific to a particular farm. It is also possible, in case of large farms, to use sub-PBox that will serve only a subset of the farm, or in case of a large Grid, to have national or regional PBoxs that will be authoritative for only a subset of the Grid itself.

Each and every client that wants to be policy-aware (RB, CE, SE, etc...), also known as a Policy Enforcement Point (PEP)[**?**] has a configured PBox that will be contacted whenever a policy decision is required. This PBox will at this point make its own decision and communicate it back to the resource.

This behaviour assure the robustness of the architecture. Even if all the rest of the grid becomes unreachable, a farm is still able to function. However, this also outlines a simple need: a Farm PBox must have knowledge of all the applicable policies, whether they be local, farm or VO policies. For this reason, a PBox can send new policies to the PBoxes at other layers, and those PBoxes may then behave in one of two different ways:

1. If the sender PBox is known as a *peer*, then the received policies are put in a waiting queue until the local administrator reviews them and decides whether to accept or refuse them.

2. If the sender PBox is known as a *master*, then the received policies are immediately accepted.

The decision is then communicated to the sender PBox, and if the policies were accepted they become immediately valid and are communicated to lower level PBoxes.

By default, PBoxes consider other PBoxes *peers*, and so a very important goal is achieved. They permit uniform behaviour among themselves (via the hierarchical communication) and leave total control on the local administrators (via the requirement of an esplicit administrative authorization).

The *master* settings exist because in some situations, like a very large farm, it makes sense to have a multiplicity of PBoxes to lighten the load, while keeping one as the central administration point.

A PBox View

The PBOX is the basic building block of this infrastructure. As can be seen from picture **??**, a PBox itself is composed by several modules:

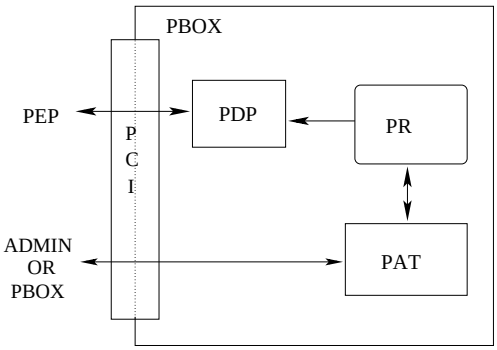

Figure 2: Architecture of G-PBox

PDP The Policy Decision Point (PDP) is the module that receives requests for policies and sends back responses. It uses the XACML 1.0 language to express both policies and requests/responses. This allows us to use custom adaptations of standard and well-proven software tools for this part of the architecture.

PAT The Policy Administration Tool (PAT) is the module that the administrator of a PBox normally uses. It is used to create, review and send policies, and to take decisions on policies received from other PBoxes.

PR The Policy Repository (PR) is the module that stores all the policies, both locally-created or received from remote, along with such information as status (accepted, refused), origin (from which PBox it originated), etc...

PCI The Policy Communication Interface (PCI) is a layer that surrounds the PBox and is used for all its communications with the outside, whether they be other PBoxes or PEPs. The communication itself may be unencrypted or protected by GSI, depending on the configuration. By default, communications between PBoxes are secure, confidential and mutually authenticated, while communication between a PBox and a PEP is unencrypted.

A Resource View

From the resource point of view, the G-PBox appears as a black box, and there is no knowledge of the connections between PBoxes. Indeed, there is no knowledge of the whole structure of G-PBox.

The resource needs only to implement a PEP, to which it will delegate the policy related tasks. It will be the PEP's responsibility to contact its PBox to send requests, to obtain responses and to translate them in a format its resource can understand, while the resource itself should be capable of interpreting these resource and acting based upon them.

The policies

Policies are defined in the XACML[**?**] language. However, they are included as part of a PBOXPolicy schema that also associates to them additional informations: an internal status (accepted, rejected, pending), an originator and a set of couples (PBox, status) that record the status of the policy on the peer PBoxes to which it has been sent.

Extensions of XACML XACML, as defined by Oasis, sets its scope to policy decision on authorization issues. However, this is not a problem in practice because, using the concept of Obligation and some wisdom, the language can be used to implement all kind of policies, even management ones. As an (informal) example, the policy: *Infngrid jobs should be moved to the igrid queue* can be expressed, minus the exact syntax, like: *Infngrid can submit jobs here, but (Obligation) move them to the igrid queue*.

It is clear though that there is a need for additional actions, and indeed we defined (and are still defining, based on actual needs) a new set of actions, of which one representative is: *uri:pbox:1.0:submit*.

SCALABILITY EXAMPLE

The G-PBox architecture best shows its capability with complex multi-layered structures.

Figure **??** shows also another example of scalability of the PBox policy system. In many cases, a site can be extremely complex. So it's not rare to divide the site in several sub-sites. The figure shows $FARM_1$, composed by two sub-farms ($SubFARM_1$ and $SubFARM_2$) with their own PBoxes placed in a layer lower than the $FARM_1$ PBox.

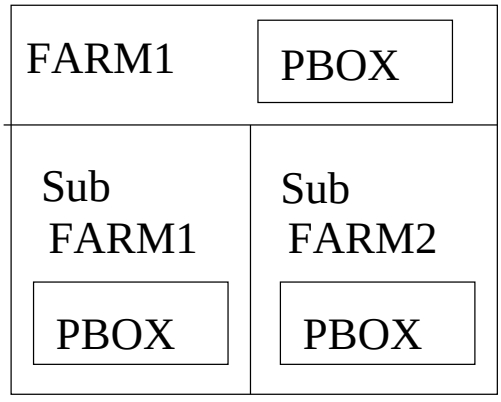

Figure 3: Grid scalability

It would also be possible to "join" two different grids, while keeping them separate, by sharing some of their respective resources. This can be done by exhanging relevant policies between the two grid-PBoxes, it becomes possible for users belonging to one of the two grids to access a subset of resources belonging to the other grid according to the exchanged policies.

OPEN ISSUES

There are at least three problems that are still open:

- There are situations where to properly evaluate policies about resource consumption on the whole grid, it is necessary to have a global view of the grid. E.g. "Allow the job if the user has run less than a total of 50 jobs." To deal with this kind of policies a working accounting system is necessary.
- In case of heavy policy exchange on the system, since administrators need to explicitly accept or refuse a new policy, they also need to be particularly responsive. A foreseen workaround is to configure the sending PBoxes a *master*.
- Since not all VO-wide policies need to be known by the local sites, by direct submission to them it may become possible to bypass them by directly submitting a job. There are two possible solutions: 1) to distribute VO-wide policies to local sites, so that they are aware of them, and 2) to require job subsmission through some kind of policy-aware submission Broker.

CONCLUSIONS AND ACKNOWLEDGEMENTS

While the system described in this paper seems to be capable of tackling this problem, development and testing is still ongoing. A prototype version is available at http://infnforge.cnaf.infn.it/projects/pbox

The authors wish to thank the EGEE and Grid.IT projects for their support and funding of this work.

REFERENCES

[1] L. Pearlman, V. Welch, I. Foster, K. Kesselman and S. Tuecke, A Community Authorization Service for Group Collaboration, IEEE Workshop on Policies for Distributed Systems and Networks, 2002.

[2] Guide to LCAS: http://www.dutchgrid.nl/DataGrid/wp4/lcas/edg-lcas-1.0.3/lcas.html

[3] M. Lorch, D. Adams, D. Kafura, M. Koneni, A. Rathi, S. Shah, The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments, 4th Int. Workshop on Grid Computing - Grid 2003, 17 November 2003 in Phoenix, AR, USA

[4] Architectural design and evaluation criteria: WP4 Fabric Management, DataGrid-04-D4.2-0119-2-1, 2001.

[5] The Oasis eXtensible Access Control Markup Language TC, eXtensible Access Control Markup Language (XACML) Version 1.1, http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf

[6] R. Yavatkar, D. Pendarakis, R. Guerin, A Framework for Policy-based Admission Control, RFC 2753,

[7] GACL — A Grid ACL manipulation Library, http://www.gridpp.ac.uk/authz/gacl/