New tools and technologies for the ENEA-GRID pervasive integration

Summary

development of innovative GRID tools and solutions oriented to the improvement of the management and functionality of ENEA-GRID infrastructure, in the framework of the CRESCO project.
The ENEA-GRID environment is based on two fundamental pillars, the Andrew File System (OpenAFS , a distributed file system providing facilities for the resource distribution over a WAN), and the Load Sharing Facility (LSF Multicluster, a software capable to handle the resource and load management in a multi-site environment).
The poster will present three innovative projects that have been engineered within CRESCO with the aim to be fully integrated with the main ENEA-GRID components.

(1) **AMACA** (AFS Memorize and Check Application) is a two-module application aiming to trace the status of every AFS core component, while providing an open, state-of-the-art monitoring platform for both administrators and users. One of the main AMACA succesful goals is to expose a set of APIs (already adopted by other supporting tools) for the native web-based authentication towards AFS.

(2) **ARCO** (AFS Remote Command Operator) is a Service Oriented web-based application, whose purpose is to submit remote commands to machines registered by system administrators and has been developed specifically to optimize the management of the LSF daemons of the computational nodes. The application make use of the AFS authorization mechanism to grant the desired authority to selected users.

(3) **The integration of the OpenAFS ACL and the Apache Web Server** operation, so that ENEA-GRID users can easily publish on the Web selected portions of their AFS data space, while keeping control on the access mechanism. This problem has been solved with a set of open, standard tools whose interoperability results in a native interface between AFS and the Apache web server.

1 AMACA (AFS Memorize And Check Application)

CRAWLER – The crawler module of AMACA (based on N. Gruener's AFS APIs) is a Perl application aiming to check the status of every AFS core component. The indexing results are handled by a memorization module, that is responsible to store the history of filesystem's events in a MySQL backend (making easier to perform subsequent fine-grained data mining operations). Every crawler invocation is differentiated by the others by an unique ID called **snapshot**. While the crawler can be invoked both directly (i.e: a user belonging to a certain PTS group explicitly runs it via the web Explorer module) or not (i.e.: its execution is scheduled in the O.S.'s cron), also the database is splitted in two: a "scratch" portion is dedicated to the latter execution kind, so that the administrator can solve problems and see "instantly" what is the effect of his intervention without interfering with the global informations.

EXPLORER - The explorer module of AMACA is a web 2.0 compliant application, written in Php with some AJAX elements, providing a comfortable interface to the analysys of the crawling results. The Explorer module provides facilities both for the visualization of "static data" (i.e.: size and number of volumes and partition, sync site, alarms about critical events, …) and interactive queries. The Explorer module is adaptive: every shown data can be shrunk to a particular snapshot/domain in order to focus the attention only on situations detected locally.

Authentication Mechanism

AMACA exposes a set a well-structured interface for the user web authentication over OpenAFS.
This API realizes an Inter-Process Communication mechanism with the PAG shell, so that every protected web application is:

• **ATOMIC** – Using PAG shell allows to have multiple, concurrent user requests for a valid couple ticket/token. The Pag Shell provides an "user-aware" isolation level for an object (the token) whose existance, otherwise, would be limited to one instance at a time (generating harmful race conditions).

• **SECURE** – The needed user data are stored in a cookie, that is encrypted with strong algorithms provided by the open source *mcrypt* library. This feature can be enforced if the web application is configured to run under an SSL (https) tunnel (just as we do in our production case).

• **INTEGRATED** – The user credentials are verified in the AFS native way.

2 ARCO (AFS Remote Command Operator)

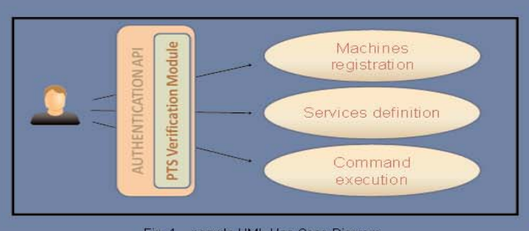

Fig. 1 – pseudo-UML Use Case Diagram

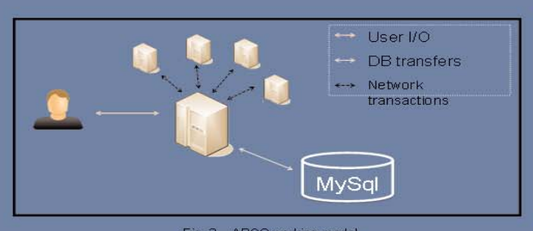

Fig. 2 – ARCO working model

ARCO is an application engineered in order to execute commands on remote machines. While its original purpose foresaw to be fully integrated with (and maybe focused on) LSF multicluster platform, it has become soon a Service Oriented application (i.e. a software capable to handle indifferently potentially any remote tool).
With ARCO (fig. 1), an administrator can register the services he wants to deal with, the machines where the services are, and perform the same execution in a visual, convenient, secure way.
Currently it is possible for the administrator to register the target machines by writing them in a text file, or, better, feeding ARCO with the LSF configuration files (so, every commented entry will be considered registered but unable to receive commands). We plan also to support many other file formats (PBS,…)
Every task ARCO performs is logged (fig. 2) in a MySQL backend (where also the associations machine/service are stored).
ARCO uses, on the authentication side, the same APIs than AMACA. In this case, anyway, they have been extended in order to fully support a (customizable) set of PTS memberships (because, to increment the flexibility and the security, not any operation can be performed by every group). Such checking is delegated to an ad-hoc module, depending from the first "classical" one, who realizes a two-way authentication mechanism.

3 Apache and AFS integration

The work on the integration between the Apache web server and openAFS has involved different tasks, referring to three main scenarios:
1.The common scenario (fig.3) regards the way we want our web server to publish the pages of users, projects and softwares. While OpenAFS provides a robust sharing environment where the resources are stored, it is convenient and efficient to simply link them in the Apache's document root in order to have them instantly on the Internet. A set of scripts (which will be soon available for download) has been implemented in order to have a full (incrementally filled) list of all available published pages.

2.In some cases a project/software administrator or, generally speaking, a resource owner, wants his data to be accessible only by selected users or group (fig. 4). It could seem an easily solvable task, if using the authentication APIs. Unfortunately the latter ones cannot be useful, for several reasons:

- A minimal customization/knowledge is required in order to set up the necessary working configuration
- They assume that the calling pages will embed the necessary code to perform control checking over a cookie (whose presence is mandatory in order to successfully make access to the protected pages): this requires at least the presence of an Index file, who is frequently missing (i.e.: users often want to share only list of documents, published "as is" with no format at all)

We needed a "general purpose" solution that supported also the native AFS's protection mechanisms (I.e.: Access Control Lists over protected directories). Such a requirement didn't map any existent solution, if applied to Apache 2.2.
To achieve the goal, we have engineered a patch for the Jan Wolter's mod_authnz_external apache module (aiming to bypass the built-in apache authentication mechanism, in order to get it tailored on the user needs), so that it could export the current URI's physical path. Such patched Apache extension has been coupled with two other software modules, performing the control over the requestor's membership and mapping the results on the current directory rights. The only issue we have found is related to the slightly increasing response time: Apache needs to call the interpreter for every step of the directory walking and this results in a little overhead.
The working model of the system is shown in figure 5: an user initiates an https request via his browser in order to get authorized toward a protected AFS directory (notice that having SSL support is mandatory: every plain http request made over a secured folder results in a 500 Internal Server Error message). The httpd's environment is trasnferred, via the mod_authnz_external module) to two interoperating pieces of software, providing, respectively, the credentials checking (over the native AFS protection/kerberos servers) and the group control .
Also in the present case, just as we had to do during the implementation of the AuthAPIs, we needed to deal with a PAG shell environment, in order to guarantee the concurrency support and to avoid race conditions.

3.The third consideration has been done about the securization of the new web server system: the "symlink-aware" structure could be harmful: an incautious user could create links toward sensitive system files without any control. To address this potential issue, we moved to three directions:

- Forcing Apache to follow the link only if its owner matched with the one of the link's target (system files are usually owned by root)
- Filling in the passwd file with the AFS user names, so that it could be possible to chown every linked page (in the document root) to its AFS owner
- Allowing only root to be accepted when receiving a remote access request(via SSh).

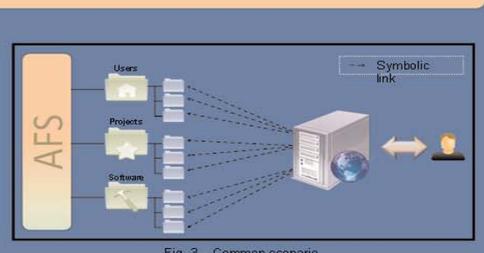

Fig. 3 – Common scenario

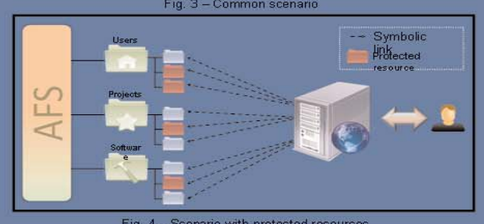

Fig. 4 – Scenario with protected resources

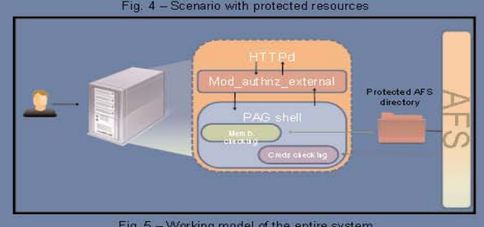

Fig. 5 – Working model of the entire system

Alessio Rocchi [1], Giovanni Bracco, Andrea Santoro, Carlo Sciò [2]
[1] CASPUR (Consorzio Applicazioni Supercalcolo per Università e Ricerca), Via dei Tizii 6, 00185 Roma – [2] Esse3Esse
ENEA C.R. Frascati – Via E. Fermi 45, 00044 Frascati (Roma)