An Authorization Architecture Oriented to Engineering and Scientific Computation in Grid Environments

Changqin Huang [1,2], Guanghua Song[1,2], Yao Zheng[1,2], and Deren Chen[1]

[1] College of Computer Science, Zhejiang University, Hangzhou, 310027, P. R. China
[2] Center for Engineering and Scientific Computation, Zhejiang University, Hangzhou, 310027, P. R. China
{cqhuang, ghsong, yao.zheng, drchen}@zju.edu.cn

Abstract. Large-scale scientific and engineering computation is normally accomplished through the interaction of collaborating groups and diverse heterogeneous resources. Grid computing is emerging as an applicable paradigm, whilst, there is a critical challenge of authorization in the grid infrastructure. This paper proposes a Parallelized Subtask-level Authorization Service architecture (PSAS) based on the least privilege principle, and presents a context-aware authorization approach and a flexible task management mechanism. The minimization of the privileges is conducted by decomposing the parallelizable task and re-allotting the privileges required for each subtask. The dynamic authorization is carried out by constructing a multi-value community policy and adaptively transiting the mapping. Besides applying a relevant management policy, a delegation mechanism collaboratively performs the authorization delegation for task management. In the enforcement mechanisms involved, the authors have extended the RSL specification and the proxy certificate, and have modified the Globus gatekeeper, jobmanager and the GASS library to allow authorization callouts. Therefore the authorization requirement of an application is effectively met in the presented architecture.

1 Introduction

Grid Computing [1] emerges as a promising paradigm for coordinating the sharing of computational and data resource and wide-area distributed computing across organizational boundaries. The sharing of code and data on the grid gives rise to many great challenges. Grid infrastructure software such as Legion [2] and Globus [3] enables a user to identify and use the best available resource(s) irrespective of resource location and ownership. However, realizing such a pervasive grid infrastructure presents many challenges due to its inherent heterogeneity, multi-domain characteristic, and highly dynamic nature. One critical challenge is providing authentication, authorization and access control guarantees.

Among relevant grid applications, due to the capability of full utilization of many valuable resources, engineering and scientific computing is suited for being solved in grid environments. This type of task is commonly either computation-intensive or

P.-C. Yew and J. Xue (Eds.): ACSAC 2004, LNCS 3189, pp. 461–472, 2004.

data-intensive, the problem granularity is widely large and computational tasks are often long-lived. It needs be divided into many subtasks, and then be distributed to many relevant nodes and run in parallel, and the management of subtasks is dynamic. The issue needs not only fine-grained authorization for resource usage and management but also fine-grained authorization for task management to meet the needs of this type of application.

In this paper, we focus on the security requirements posed by engineering and scientific computation applications in grid. We present the Parallelized Subtask-Level Service Authorization (PSAS) architecture for fine-grained authorization policies and enforcement mechanism for both resource usage/management and task management. The context-aware authorization is exercised by mapping a community member to a multi-value community policy and adaptive transition, and a delegation mechanism collaboratively performs task management together with a relevant management policy. It enforces these mechanisms to enable fine-grained authorization based on Globus Toolkit version 2.2.

This paper is organized as follows: Section 2 reviews background and related work in the arena of grid security. In section 3, the proposed authorization architecture and overall policy are described. Context-aware authorization is presented in Section 4. Section 5 describes the current implementation of the architecture within Globus. Finally, conclusions and future work are addressed in Section 6.

2 Backgrounds and Related Work

2.1 Authorization in Grid Middleware

As the rapid advancement of the grid researches and applications, diverse grid middlewares are widely developed and deployed. At present, there are three main pieces of grid middlewares, Globus [3], Legion [2], and UNICORE [17]. The Globus toolkit is the most popular grid environment and the de facto grid standard. However its current security services are yet poor, for example: use of static user accounts, coarse granularity, and application dependent enforcement mechanisms. Globus has adopted the Grid Security Infrastructure (GSI) [5] as the primary authentication mechanism. GSI defines single sign-on algorithms and protocols, cross-domain authentication protocols, and temporary credentials called proxy credentials to support hierarchical delegation [6]. Main weaknesses of the Globus security services are described as follows:

1. Globus deals with all privileges of subtask irrespective of the privilege difference among its subtasks. That is, after the simple authentication is exercised, the resource allows the task to use all privileges of the user; similarly, so do subtasks run in parallel. It violates commonly the least privilege principle [7].
2. The issues of the context-aware community policy are not concentrated on, so the authorization of resource usage and task management is not flexibly characterized by the community. The scenario is not suited for large-scale wide-area collaboratively scientific computation in Virtual organization.

3. In Globus, normally, task management is only the responsibility of the users who have submitted the job. Due to the dynamic environment and the long-lived feature of engineering and scientific computation, this coarse-grain authorization for task management cannot meet the need of agile job management in VO.

2.2 Related Work

In recent years, many grid security issues (architectures, policies and enforcement mechanisms, etc) have been researched. And the related researches are making great progress. Among many related works, main researches are presented in the following:

I. Foster et al. [5] provide the basis of current grid security: "grid-map" mechanism, mapping grid entities to local user accounts at the grid resources, is a common approach to authorization. A grid request is allowed if such a mapping exists and the request will be served with all the privileges configured for the local user account. Obviously, these authorization and access control mechanisms are not suitable for flexible authorization decision.

L. Pearlman et al. [8] propose the Community Authorization Service (CAS) architecture. Based on CAS, resource providers grant access to a community accounts as a whole, and community administrators then decide what subset of a community's rights an individual member will have. Drawbacks of this approach include that enforcement mechanism does not support the use of legacy application, that the approach of limiting the group's privileges violates the least-privilege principle and that it does not consider authorization issue of task management.

W. Johnston et al. [9] provide grid resources and resource administrators with distributed mechanisms to define resource usage policy by multiple stakeholders and make dynamic authorization decisions based on supplied credentials and applicable usage policy statements. This system binds user attributes and privileges through attribute certificates (ACs) and thus separates authentication from authorization. Fine-grained access decisions are enforced via such policies and user attributes. However, It does not provide convenience for the use of legacy applications, and does not consider authorization issue of task management.

R. Alfieri et al. [10] present a system conceptually similar to CAS: the Virtual Organization Membership Service (VOMS), which also has a community centric attribute server that issues authorization attributes to members of the community. M. Lorch et al. [11] give the same architecture, called PRIMA. Except that in PRIMA the attributes are not issued by a community server but rather come directly from the individual attribute authorities, PRIMA and VOMS have similar security mechanisms. They utilize expressive enforcement mechanisms and/or dynamic account to facilitate highly dynamic authorization policies and least privilege access to resources. However, they do not consider authorization issue of task management, and in their study, the overhead of authorization management is larger. They only support the creation of small, transient and ad hoc communities.

Besides the typical paradigms mentioned above, M. Lorch et al. [12] enable the high-level management of such fine grained privileges based on PKIX attribute cer-

tificates and enforce resulting access policies through readily available POSIX operating system extensions. Although it enables partly the secure execution of legacy applications, it is mainly oriented to collaborating computing scenarios for small, ad hoc working groups. G. Zhang et al. [13] present the SESAME dynamic context-aware access control mechanism for pervasive Grid applications by extending the classic role based access control (RBAC) [14]. SESAME complements current authorization mechanisms to dynamically grant and adapt permissions to users based on their current context. But, monitoring grid context in time is high-cost. K. Keahey et al. [15] describe the design and implementation of an authorization system allowing for enforcement of fine-grained policies and VO-wide management of remote jobs. However, it does not specify and enforce community policies for resource usage and management currently. S. Kim et al. [16] give a WAS architecture to support a restricted proxy credential and rights management by using workflow. It does not consider the actual conditions of large-scale task running at many nodes in parallel, the large overhead of fine-grained division of task and associated authorization confine its application to a limited area.

3 PSAS Architecture

3.1 PSAS Architecture Overview

PSAS architecture is concerned with the different privilege requirements of subtasks, user privilege of resource usage and resource policy in virtual community, task management policy and task management delegation. So PSAS architecture includes three functional modules and a shared enforcement mechanism as shown in Figure 1.

To minimize privileges of a task, the parallelizable task is decomposed and the least privileges required for each subtask is re-allotted after analyzing the source codes of the task. This contribution is described in the part of Subtask-level authorization module in the next sub-section. To apply a flexible task management, a delegation mechanism collaboratively performs the authorization delegation for task management together with a relevant management policy. Its details exist in the part of Task management authorization module in the next sub-section. A context-aware authorization approach is another contribution based on PSAS architecture, and it is presented in Section 4.

3.2 Privilege Management and Overall Authorization Policy

Besides the shared enforcement mechanism, in PSAS, there exist three modules to implement Privilege management and overall authorization policy: Subtask-level authorization module, Community authorization module, and Task management authorization module.

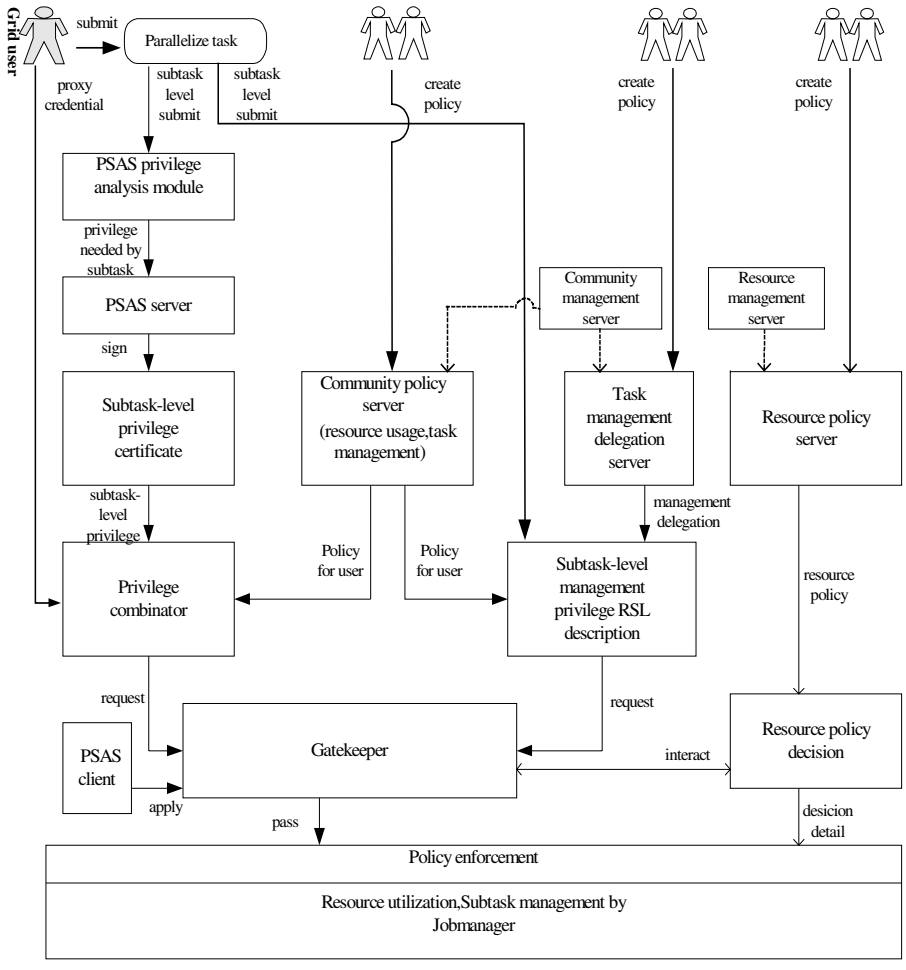


Fig. 1. PSAS architecture overview.

Subtask-level authorization module concentrates on minimizing privileges of tasks by decomposing the parallel task and analyzing the access requirement. To further conform to the least-privilege principle, a few traditional methods restrict the privileges via the delegation of users themselves rather than the architecture, moreover, they only restrict privileges of a whole specific task, not for its constituents (such as subtask). In engineering and scientific computation application, a task is commonly large-scale. It need be divided into many subtasks, and then be distributed to many relevant nodes and run in parallel. Whilst, even though the task is the same, privileges required for distinct subtasks may differ according to operations of these subtasks. By the parallelization and analysis of the task, PSAS can obtain the task's subtasks and relevant required privileges: subtask-level privilege pair. The privileges indicate the information about associated subtask required access to resources at certain nodes. Each task has a subtask-level privilege certificate for recording subtask

and privilege pair. An example of this certificate is shown in Figure 2. To prevent malicious third party from tampering with a subtask-level privilege certificate, a trusted third party (PSAS server) signs the certificate. To control a subtask's process and verify the subtask-level privilege certificate, PSAS client will run during resource utilization.

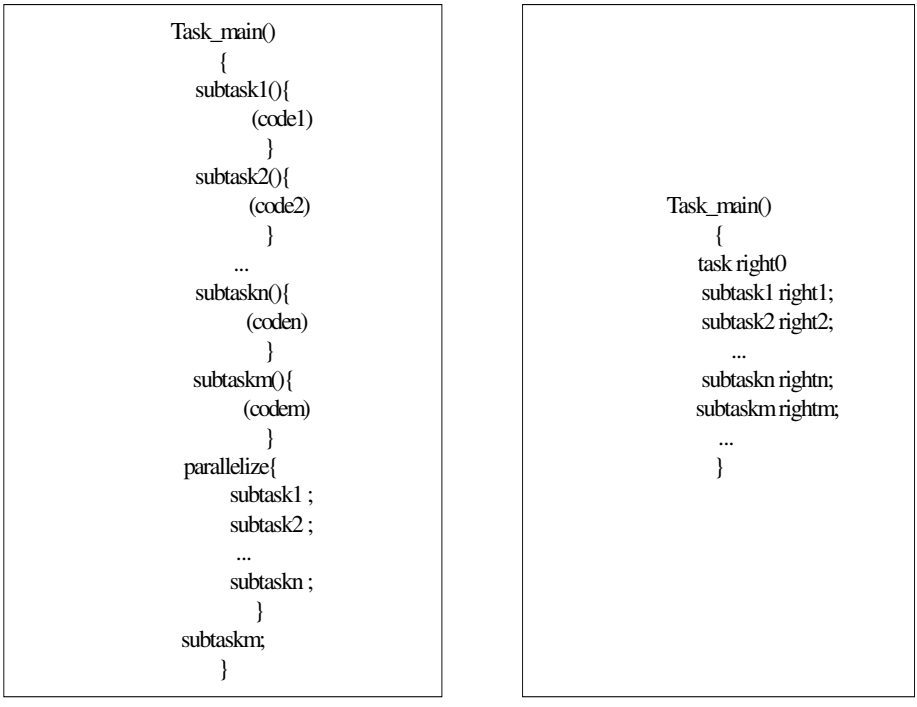


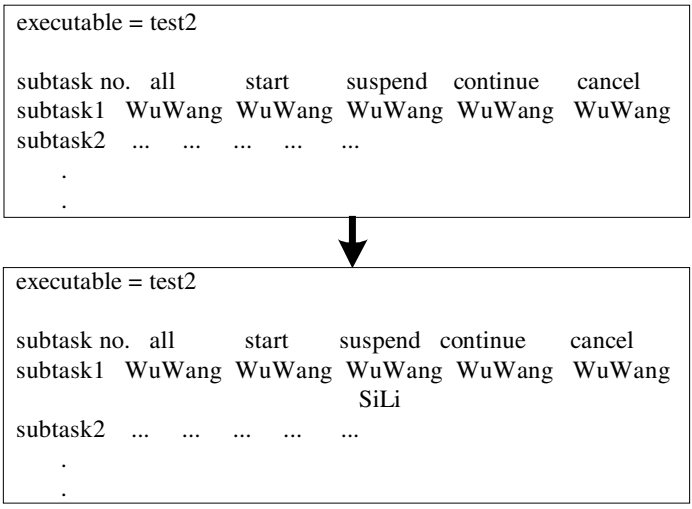
Fig. 2. An example of task and subtask-level privilege certificate.

Subtask-level authorization module contains PSAS privilege analysis module, PSAS server, privilege combinator (shared by community authorization mechanism) and PSAS client.

Community authorization module addresses community authorization mechanism for community member. In Globus, "grid-map" mechanism is conducted, but it is neglected that grid collaboration brings out common rules about privilege for resource usage, resource permission, and so forth, which makes grid security fall into shortage of adequate availability. PSAS architecture imposes similar CAS [8] mechanism with a combination of traditional grid user proxy credential and CAS, as well as the task management policy is added into the community policy server. Two trusted third parties (a community management server and a resource management server) and two policy servers (a community policy server and a resource policy server) are exercised. A community management server is responsible for managing the policies that govern access to a community's resources, and a resource management server is responsible for managing the policies that govern resource permission to grid users.

Two policy servers store the policy for community policy and resource policy respectively. The ultimate privileges of a grid user are formed by the relevant proxy credential and the policy for this user, and the actual rights need accord with resource policy by resource policy decision module during policy enforcement. The policy servers are built or modified by the community administrators or certain specific users.

Community authorization module is composed of a community management server, a resource management server, a community policy server, a resource policy server, privilege combinator, and resource policy decision module.




```
/from:/O=Grid/O=Globus/OU=OU=zju.edu.cn/CN=WuWang/CNType= user
/to:/O=Grid/O=Globus/OU=OU=zju.edu.cn/CN=SiLi/CNType= user
&(action = suspend)(executable = test2)(subtask = subtask1)(directory = /tmp/test2)
```

Fig. 3. A task management delegation and the relevant change of the subject list.

```
/O=Grid/O=Globus/OU=OU=zju.edu.cn/CN=WuWang/CNType= user:
&(action = start)(executable = test1)(subtask = subtask1)(directory = /tmp/test1)(count<2)
&(action = start,suspend,cancel)(executable = test2)(subtask = subtask1)(directory = /tmp/test2)(count<4)

/O=Grid/O=Globus/OU=OU=zju.edu.cn/CN=Grid administrator/CNType= group:
&(action = all)(executable = all)(subtask = all)(directory = /tmp)
```

Fig. 4. An example of task management description.

Task management authorization module is responsible for managing privilege of task management, authorization to task management and related works. In dynamic

grid environments, there are many long-lived tasks, for which static methods of policy management are not effective. Users may also start jobs that shouldn't be under the domain of the VO. Since going through the user who has submitted the original job may not always be an option, the VO wants to give a group of its members the ability to manage any tasks using VO resources. This module imposes a community task management policy and task management delegation to describe rules of task management, and both of the two mechanisms are beneficial to flexible task management in an expressive way. A community task management policy denotes the rules of task management in the whole community, and it is combined into the community policy server. Task management delegation server records a variety of management delegation relations among community users. Once the delegation relation is formed, this task will be able to be managed by the delegate user at its runtime. A community management server is responsible for authenticating task management delegation. To keep compatibility with special task management, the subject list only consists of the owner of the task by default. Figure 3 shows a task management delegation and the change of the subject list for task management privilege to this delegation. Subtask-level privilege management RSL description server produces the task management description, which is expressed by extending the RSL set of attributes. An example of task management description is shown in Figure 4.

Task management authorization module includes a community task management policy, task management delegation server and Subtask-level privilege management RSL description server.

4 Context-Aware Authorization

Based on the PSAS Architecture, the dynamic authorization is able to complement with a low overload; meantime, little impact is enforced on grid computation oriented to scientific and engineering. The main idea is that actual privileges of grid users are able to dynamically adapt to their current context. This work is similar to the study in the literature [13]; however, our context-aware authorization is exercised by constructing a multi-value community policy. The approach is completed according to the following steps:

1. Rank the privileges belonging to each item in a traditional community policy. We divide the privileges of a community member into three sets of privileges: Fat Set, medium Set and thin Set. Fat Set is rich set with the full privileges of this community member, and is suited for the best context at runtime; for example, at some time, the context is fully authorized nodes with least resources utilized, then the node will be able to provide its user most privileges. Medium Set is a set of privileges decreased, and Thin Set is the least privilege set for the community member. When Thin Set is enforced, the subtask belonging to this community member will use the less resources (i.e. less memory, less CPU cycles, etc) or be canceled.
2. Construct a multi-value community policy. After finishing the above, we must rebuild the community policy. To keep compatible with the traditional policy, we

only add two sub-items below each item, and the two sub-items are inserted for medium Set and thin Set, respectively. We apply the policy language defined by the literature [15], and introduce two new tags "$1" and "$2" as the respective beginning statement. An example of a policy of resource usage and task management is shown in Figure 5. The statement in the policy refers to a specific user, Wu Wang, and in the first item, it states that he can "start" and "cancel" jobs using the "test1" executables; The rules also place constraints on the directory "/usr/test1" and on the count "<4". In its multi-value community policy, corresponding Fat Set maps to the first item without the change of privileges, its Medium Set and Thin Set respectively map to the next two sub-items below the first item, and their authorizations are changed. For instance, in its corresponding Thin Set, the empty statement prevents him from doing any control using the "test1" executables, the statement "memory<20" places a constraint of used memory <20M, and the statement "directory=/tmp/test1" places a constraint of resource usage.

```
/O=Grid/O=Globus/OU=OU=zju.edu.cn/CN=WuWang/CNType= user:
&(action = start,cancel)(executable = test1)(subtask = subtask1)(directory = /usr/test1)(count<4)
&(action = start,suspend,cancel)(executable = test2)(subtask = subtask2)(directory = /usr/test2)(count<6)
```

```
/O=Grid/O=Globus/OU=OU=zju.edu.cn/CN=WuWang/CNType= user:
&(action = start,cancel)(executable = test1)(subtask = subtask1)(directory = /usr/test1)(count<4)
 $1(action = cancel)(executable = test1)(subtask = subtask1)(directory = /usr/test1)(count<2)(memory<100)
 $2(action = )(executable = test1)(subtask = subtask1)(directory = /tmp/test1)(count<2)(memory<20)
&(action = start,suspend,cancel)(executable = test2)(subtask = subtask2)(directory = /usr/test2)(count<6)
 $1(action = start,cancel)(executable = test2)(subtask = subtask2)(directory = /usr/test2)(count<4)
 $2(action = cancel)(executable = test2)(subtask = subtask2)(directory = /tmp/test2)(count<2)(memory<20)
```

Fig. 5. A traditional policy and its corresponding multi-value community policy.

3. The context-aware authorization is conducted via the above multi-value community policy at runtime. As shown in Figure 6, the model uses a Context Agent as an entity to sense the context information. The Transition Controller accepts the trigger from associated Context Agent and makes a decision of transition of privilege set. In addition to these common policies, the Community Policy Server contains transition policy, as a rule of state transition, and event policy, as a rule of sense event. For example, when the memory of hosting node becomes exhausted, the event notifies Context Agent and let it sense and trigger the Transition Controller.

5 Enforcement Mechanisms

Enforcement of fine-grained access rights is defined as the limitation of operations performed on resources or tasks/subtasks by a user to those permitted by an authoritative entity. Based on the Globus Toolkit 2.2, PSAS architecture implements the subtask-level authorization, flexible task management, and context-aware authorization.

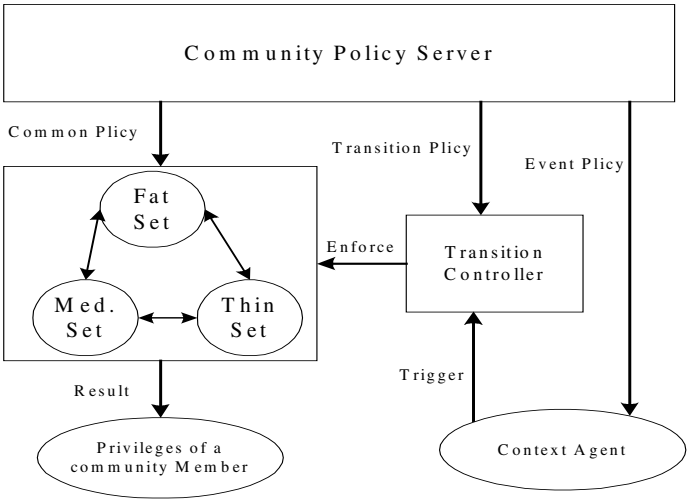


Fig. 6. The context-aware authorization model.

To implement task management authorization module, PSAS creates a job management controller- a component by extending jobmanager in GRAM. When the jobmanager parses users' job descriptions, the job management controller parses and evaluates subtask-level privilege certificate, and makes a decision of certain task management permission. The job management controller integrates with the jobmanager by an authorization callout API. The callout passes the relevant information to the job management controller, such as the credential of the user requesting a remote job, the action to be performed (such as start or cancel a job), a unique job identifier, and the job description expressed in RSL. The job management controller responds by the callout API with either success or an appropriate authorization error. This call is made whenever an action needs to be authorized; that is, it happens before creating a job manager request, and before calls to cancel, query, and signal a running job. PSAS extends the GRAM protocol to return authorization errors with reasons of authorization denial as well as authorization system failures. All of task management delegations and community policies are described in complex task management cases, and they are translated into a RSL regular description by Subtask-level privilege management RSL description server.

To enforce subtask-level authorization module, PSAS employs the proxy certificate with an extension field in the form of standardized X.509 v2 attribute certificates, and makes signed subtask-level privilege certificate embedded into the proxy certificate's extension field. At the same time, PSAS architecture modifies the Globus gatekeeper and jobmanager to put subtask-level authorization into practice. For PSAS client needs to manage a subtask's process and checks whether the subtask is running according to the subtask-level privilege certificate, the GASS library is modified to communicate with PSAS client, and the relevant callout APIs are created. After mutual authentication between a user and a resource, the resource obtains subtask-level privilege certificate located in the proxy certificate's extension field. Then the sub-

task-level privilege certificate is verified and finds out job identifier. Finally, PSAS client guarantees a subtask's process according to the subtask-level privilege certificate by interacting with GASS.

Community authorization module in PSAS is executed as in CAS [8]. The GSI delegation feature is extended to support rich restriction policies in order to allow grantors to place specific limits on rights that they grant. PSAS employs extensions to X.509 Certificates to carry out restriction policies. However, there exist some differences between CAS and PSAS in the community authorization module. That is, the CAS uses restricted proxy credentials to delegate to each user only those rights granted by the community policy; but the latter regards the ultimate privileges as a privilege combination of the "restricted proxy credentials" and the subtask-level privilege certificate. Proxy credentials are separated from identity credentials. The identity credentials are used for authentication, while the proxy credentials are used for authorization. That makes the PSAS architecture more flexible. Similar to the previous cases, the GASS library is modified to implement a policy evaluation, and the relevant callout functions are designed for call in the Globus gatekeeper. To implement the dynamic context awareness, Context Agent applies a context toolkit described in the literature [18].

6 Conclusions and Future Work

In this paper, we propose a Parallelized Subtask-level Authorization Service (PSAS) architecture to fully secure applications oriented to engineering and scientific computing. This type of task is generally large-scale and long-lived. It needs to be divided into many subtasks run in parallel, and these subtasks may require different privileges. The minimization of the privileges is conducted by decomposing the task and re-allotting the privileges required for each subtask with a subtask-level privilege certificate. With the aid of Context Agent, a multi-value community policy for resource usage and task management enables the context-aware authorization in addition to separating proxy credentials from identity credentials. The delegation mechanism collaboratively performs the authorization delegation for task management together with a relevant management policy. To enforce the architecture, the authors have extended the RSL specification and the proxy certificate and have modified the Globus gatekeeper, jobmanager and the GASS library to allow authorization callouts. The authorization requirement of an application is effectively met in the presented architecture.

At present, the PSAS architecture is only a prototype, and many issues need to be solved. So we plan, firstly, to improve the PSAS architecture in practice via complex applications, and secondly, to further study the policy based context-aware authorization of resource usage and task management based on performance metrics.

Acknowledgements. The authors wish to thank the National Natural Science Foundation of China for the National Science Fund for Distinguished Young Scholars under grant Number 60225009. We would like to thank the Center for Engineering and

Scientific Compu-tation, Zhejiang University, for its computational resources, with which the research project has been carried out.

References

1. I. Foster, C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputer Application*s, 15(3): pp.200-222, 2001.
2. A. Grimshaw, W. A. Wulf, et al., The Legion Vision of a Worldwide Virtual Machine, *Communications of the ACM*, 40(1): 39-45, January 1997.
3. I. Foster and C. Kesselman. Globus: a metacomputing infrastructure toolkit, *International Journal of Supercomputer Applications*, 11(2): 115-128, 1997.
4. S. Tuecke, et al., Internet X.509 Public Key Infrastructure Proxy Certificate Profile. 2002.
5. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, A Security Architecture for Computational Grids, *Proc. of 5th ACM Conference on Computer and Communications Security Conference*, 1998.
6. L. Kagal, T. Finin, and Y. Peng, A Delegation Based Model For Distributed Trust, *IJCAI-01 Workshop on Autonomy*, Delegation, and Control, 2001.
7. J. R. Salzer and M. D. Schroeder, The Protection of Information in Computer Systems, *Proc. of the IEEE*, 1975
8. L. Pearlman, V. Welch, et al., A Community Authorization Service for Group Collaboration, *Proc. of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks*, 2002.
9. W. Johnston, S. Mudumbai, et al., Authorization and Attribute Certificates for Widely Distributed Access Control, *Proc. of IEEE 7th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, 1998.
10. R. Alfieri, et al., VOMS: an Authorization System for Virtual Organizations, *Proc. of the 1st European Across Grids Conference*, 2003.
11. M. Lorch, D. B. Adams, et al., The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments, *Proc. of the 4th International Workshop on Grid Computing*, 2003
12. M. Lorch and D. Kafura, Supporting Secure Ad-hoc User Collaboration in Grid Environments, *Proc. of the 3rd IEEE/ACM International Workshop on Grid Computing*, 2002.
13. G. Zhang and M. Parashar, Dynamic Context-aware Access Control for Grid Applications, *Proc. of the 4th International Workshop on Grid Computing*, 2003.
14. R. Sandhu, E. Coyne, et al., Role-based Access Control Models, *Proc. of the 5th ACM Workshop on Role-Based Access Control*, 2000
15. K. Keahey, V. Welch, et al., Fine-Grain Authorization Policies in the Grid: Design and Implementation, *Proc. of the1st International Workshop on Middleware for Grid Computing*, 2003.
16. S. Kim, J, Kim, S. Hong, et al., Workflow-based Authorization Service in Grid, *Proc. of the 4th International Workshop on Grid Computing*, 2003.
17. M. Romberg, The UNICORE Architecture: Seamless Access to Distributed Resources, *Proc. of the 8th IEEE International Symposium on High Performance Distributed Computing*, 1999.
18. A. K. Dey, G. D. Abowd, The Context Toolkit: Aiding the Development of Context-Aware Applications, *Proc. of Human Factors in Computing Systems: CHI 99*, 1999.