Dynamic Creation and Management of Runtime Environments in the Grid

Kate Keahey keahey@mcs.anl.gov
Matei Ripeanu matei@cs.uchicago.edu
Karl Doering kdoering@cs.ucr.edu

1 Introduction

Management of complex, distributed, and dynamically changing job executions is a central problem in computational Grids. These executions often span multiple heterogeneous resources, cross administrative domains, and need to adjust to the changing resource availability to leverage opportunities and account for failures or policy induced failures. The executions themselves are dynamic: they often start other computations or have requirements exceeding resources that were originally allocated.

The unpredictable nature of these executions has motivated the development of tools accounting for this aspect. For example, the single sign-on capability [1] allows a Grid user to delegate rights to a computation, so that this, in turn, can start other computations on the user's behalf. Further, although initially granted for a limited time only, such delegation can be refreshed as needed. In spite of progress in these areas, however, distributed execution typically still requires the existence of static, preconfigured runtime environments such as Unix accounts. In addition to putting an undue administrative requirement on sites participating in virtual organizations (VOs) [2], this implies that accounts are statically created and do not reflect dynamically changing VO policies. We believe that runtime environments that can not only be created but also managed dynamically and automatically are essential for the development of Grids.

In this paper, we describe architecture and interfaces designed to support dynamic, secure creation and management of runtime environments (such as local Unix accounts) in the Grid based on authentication and authorization of a user. These abstractions are designed to allow users to create and manage runtime environments in a uniform manner across different technologies. We begin by describing support for individual dynamic environments and ways in which a composition of such environments can be created. We then present how we leverage these concepts in the PlanetLab testbed to build GSLab, a distributed platform for deploying and experimenting with Grid Services.

Previous work on dynamic accounts [3-5] has not presented a unified abstraction, nor has it addressed management problems in complex distributed environments like the Grid. In this paper, we lay out the abstractions and architecture for dynamic runtime environments and show how OGSI abstractions, implemented in Globus Toolkit 3 (GT3), can be leveraged.

In short, the contributions of this paper are as follows:

- We describe abstractions for creation and management of distributed, dynamic runtime environments in Grids,
- We present how features of the Open Grid Services Infrastructure (OGSI) are leveraged to implement this architecture,
- We develop GSLab a distributed, large-scale platform for deploying and experimenting with Grid Services and test these abstractions in practice.

2 Requirements and Benefits

Runtime environments allow a user to execute a computation, communicate, and store data. Local implementations might range from simple Unix accounts to sandboxing technologies [6] to virtual machines [7-9]. Regardless of their mapping to a particular technology, runtime environments should meet the following requirements:

1) *Protection*. A runtime environment should protect the user's work, as well as the resource(s) onto which it is mapped. This includes protecting resources allocated to or created by an environment from other users executing in other runtime environments. While the strongest level of protection provided might be limited by the low-level protection mechanisms available locally, users should be able to configure the level of protection used.
2) *Controlled resource usage*. A runtime environment can be mapped onto different actual resources. Resource owners should be able to control the amount of resources consumed by a runtime environment and its associated applications. In order to achieve this, a runtime environment implementation will comprise or collaborate with local resource management mechanisms.
3) *Authorization*. A runtime environment is an enforcer of well-defined authorization assertions. These policies can be dynamic and change during the lifetime of a runtime environment. For example, the CPU share or the disk space allocated to an environment might change during its lifetime. Consequently, the enforcement of these allocations should be changeable during the lifetime of the environment.
4) *Logging and audit*. Management operations on such environments need to be securely logged so that the associations between a user Grid identity and local runtime environment are available for audit.

Our objective in this paper is to describe a protocol and architecture for runtime environment creation and management in Grids. We see the following main benefits:

1) *Automating administration*. Although the Grid Security Infrastructure (GSI) [1] implements a single sign-on model, its benefits are diminished if pre-existing local accounts are required on each resource. A service that dynamically creates accounts as needed would save site administrators the burden of procuring and maintaining a static runtime environment for every user that might potentially need it. In addition, the local runtime environment can be dynamically configured based on rights granted in a specific context.
2) *Formalizing runtime environment management*. A local runtime environment can be implemented in a variety of ways. An interface for a runtime environment management would allow uniform treatment of multiple such implementations through a set of well-defined properties and thereby allow more flexibility in terms of choosing local fine-grained resource management mechanisms.

The notion of a dynamic runtime environment can be supported to some extent on all platforms, whether they provide fine-grained resource management mechanisms or not. The existence of such mechanisms should make a difference only insofar as some runtime environment properties (matching the requirements above) will, or will not be supported by a given implementation. For example, a Unix account may constitute one implementation of a dynamic runtime environment (very coarse grained one). Unix accounts may further be combined with more sophisticated schedulers (e.g., DSRT [10], SILK [11]) to add fine-grained capabilities. At the other end of the spectrum, virtual machine monitors [12] provide even better isolation and finer-grained resource allocation capabilities.

3 Design and Implementation

In this section we describe architecture and the OGSI interfaces allowing a Grid user to create, use, and manage dynamic runtime environments. We define a *runtime environment* (RTE) as a mapping from a set of rights (which could be based on a user's Grid identity) to a local runtime environment providing protection mechanisms on a resource. An RTE could be implemented as a Unix account, or a set of Unix accounts on a cluster, or it could use an implementation that offers better isolation finer granularity resource control such as a sandbox or a virtual machine.

The architecture we present leverages OGSI features such as Grid Service Handle (GSH) or soft-state lifetime management in presenting RTEs to the user. We prototyped this design using GT3.

3.1 Runtime Environment Factories (RTEFactory)

The RTEFactory implements the ogsi:Factory interface and implements the creation of RTEs. As shown in Figure 1, the createService operation should authorize the request to create an account with the requested properties. Since no standard Open Grid Services Architecture (OGSA) authorization interface currently exists our ad-hoc authorization mechanism is based on the requestor's Grid credential and a "hardcoded" callout to an access control list. An authorization failure of this operation results in an exception. On success, the RTEFactory performs the following actions: (1) securely creates an RTE implemented as a local Unix account, (2) initializes the RTE (this could for example involve ensuring that an GT3 installation is available), and (3) writes an access policy for the newly created RTE. A GSH for the service (RTEService port) representing the newly created RTE is returned.

In our GT3 implementation the creation process is securely logged using the GT3 logging mechanism. The access policy is created in the GT3 gridmap file, which creates an association between the Grid identity of the account creator and the local name of the account just created.

3.2 Runtime Environment Service (RTEService)

The RTEService represents a transient, dynamically created RTE (e.g. a dynamic account on a system). Like any other Grid Service, an RTEService is identified by a GSH.

A RTEService exposes its properties as Service Data Elements (SDE) and allows an authorized user to query and modify them. For example, if more disk space is needed than originally requested, the user can increase the quota value after obtaining proper authorization credentials from the community. Currently, we implement SDEs referring to the RTE itself (such as: terminationTime, RTE_Implementation—a non-modifiable SDEs that exposes the underlying RTE implementation: e.g. Unix account, sandbox, etc., and the local RTE name), as well as RTE's runtime properties. In the case of Unix accounts these are limited to disk quota, as other properties are not enforced. More sophisticated isolation mechanisms (sandboxing, virtual machines) will allow extending the list of SDEs to describe the CPU, memory, and network share of a RTE.

A RTEService termination is decided either automatically by the container using the terminationTime or explicitly by the user that created it. In the current implementation, RTE termination implies that the entire RTE state is cleaned: user processes are killed, user files deleted, and the gridmap file is updated appropriately. In order to facilitate management and ensure automatic cleanup we currently limit the maximum lifetime of an RTE; based on authorization privilege the user can extend it.

3.3 Interaction between the RTEService and the Resource Management System

We streamlined the interoperability between our RTE implementation and the GT3 resource management service, GRAM. The current GRAM implementation assumes that a Grid identity maps to a single user account. To overcome this limitation, we extend GRAM to accept a RTEService's GSH with a job submission. For backward compatibility, if the GSH is not presented, then the first mapping found for a Grid identity is used.

Figure 1 illustrates how dynamic RTEs may be used:

1. The user sends to the RTEFactory a request that includes the properties and lifetime for the RTE to be created, and a Grid credential. The RTEFactory authenticates the user.
2. Authorization is carried out by a policy evaluation point (PEP). If the user is authorized to create the requested RTE, then the RTEFactory instantiates it; otherwise, an authorization exception is thrown.
3. At instantiation, the RTEService creates a local runtime environment in an platform-specific way. In our case, a setuid-root program is called to create a Unix account: a local username and uid are assigned to the user, the mapping associating the Grid credential with the local username is

securely logged, and the RTE is prepared for use. Additionally, a new policy is written out allowing/restricting access to the new RTE (currently, we simply modify the grid-mapfile).

4. A GSH of the new RTEService is returned to the user.
5. The user may manipulate the properties of the RTE, such as disk quotas or termination time, by making authorized calls to corresponding operations.
6. The user can submit jobs to be run in a specific RTE by adding the GSH for the RTEService to a Globus GRAM submission.
7. When the RTEService is destroyed, the entire stated associated with the local RTE is deleted. Again, all operations are securely logged.

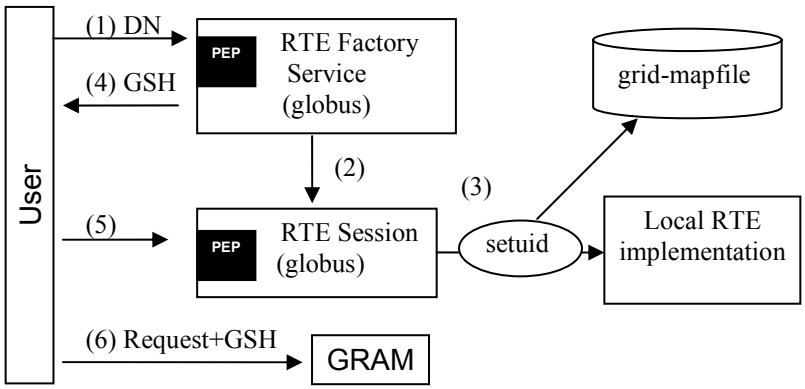

Figure 1: Runtime environment creation and management. Policy evaluation points (PES) are marked on the RTE mechanisms only; there are others within GRAM.

In order to facilitate a situation where a user does not need a RTE independent of a job execution (i.e., a RTE gets created for the duration of one execution only), we integrated the RTE creation mechanism into GRAM. In this case the resource manager (GRAM) contacts the RTEFactory directly and obtains an RTE for the user.

4 GSLab: A Platform for Experimenting with Grid Services

We used the abstractions defined above to build GSLab: a platform for deploying and experimenting with Grid Services (GS) on the PlanetLab testbed. Our goal in developing GSLab is twofold: first, to build a useful tool for the Grid community (a distributed, large-scale, controlled platform for experimenting with GS), second, to experiment with GS abstractions for managing a large-scale, widely distributed infrastructure.

PlanetLab is a general-purpose testbed for network applications spanning hundreds of nodes. The main abstraction and allocation unit currently provided by PlanetLab is a slice: a distributed set of virtual machines, one at each PlanetLab node. An environment of this scale and complexity cannot rely on statically configured accounts as RTEs. PlanetLab's current centralized slice creation and management infrastructure does not support a set of users with the dynamicity we envision for GSLab. Furthermore, in the future, slice management functionality is planned to be offloaded to independent "brokers" (and this is the space we envisage for GSLab).

GSLab builds on PlanetLab functionality: it leverages the support for multiple users while effectively isolating them and protecting resources. Further, GSLab usefulness increases if fine-grained resource allocation mechanisms are available.

Clearly, such platform benefits from the uniform RTE management abstractions described in Section 3, and for this reason we are using these abstractions here. GSLab (in fact, standard GT3 container and a few additional GS services like RTEFactory) runs in each virtual machine of a PlanetLab slice.

At a high level, GSLab allocates a distributed set of RTEs to users, then manages this set on user's behalf while dealing with all associated authentication, authorization and lifetime management issues. This set is allocated through interaction with the GSLab central administration point (AdminPoint), which manages the distributed infrastructure services (RTEFactories, basic GT3 services, gsi-sshd

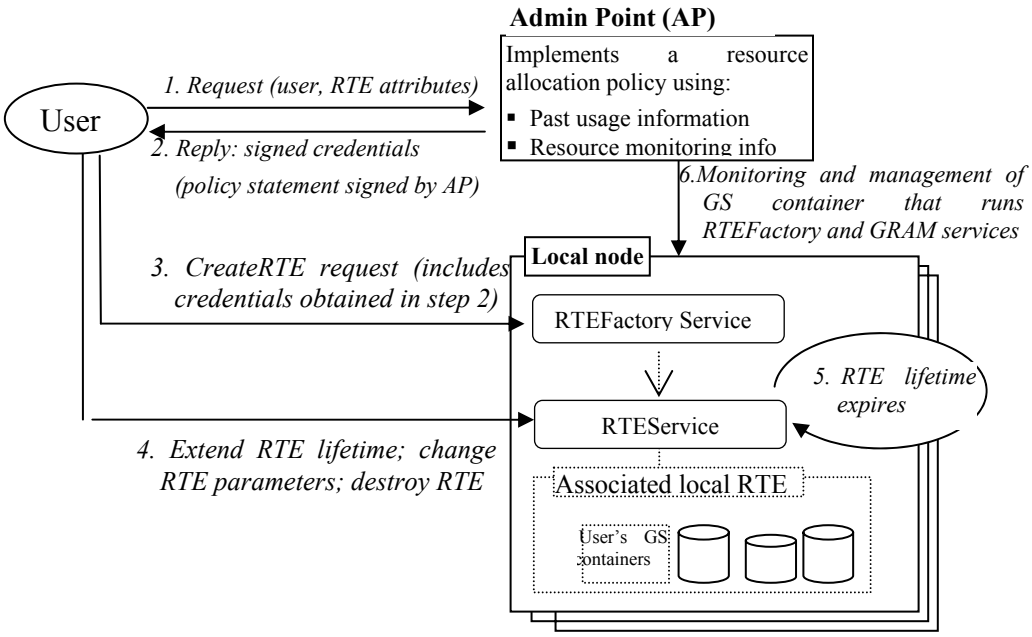

***Figure 2*:** GSLab aggregate RTE management design: Steps: (1) A user presents a request for a set of RTEs to the AdminPoint. The AdminPoint decides to allow the user to create RTEs with the desired properties on a set of nodes. (2) It returns a signed policy statement. (3) The user passes this credential to contact local RTEFactories and ask for RTEs instantiation. (4) RTEs are used through standard Globus Toolkit services: (e.g., GRAM); their attributes and lifetime can be changed if needed. (5) RTEs are destroyed either explicitly or when their lifetime expires. (6) The AdminPoint is in charge ofh monitoring and keeping essential management services (e.g., RTEFactory, GRAM) running at each node.

daemon) and serves as the central authorization point. The AdminPoint basic architecture is presented in Figure2.

RTEs provided to users are general-purpose execution environments. In addition to the RTEFactory and RTEService, GSLab provides a few utilities to facilitate experimentation and deployment of Grid Services (GS). These include scripts for GS Container management (creation/startup/shutdown) and GS deployment, together with authenticated remote login through GSI mechanisms. GSLab provides a few additional global infrastructure services: an IndexService with status information about participating nodes (where user services can publish their own information), and a global registry to bootstrap user services.

Currently we have deployed GSLab on a limited number of PlanetLab nodes. We are using this limited deployment to evaluate and fine tune GSLab. By the time of the workshop we hope to have deployed the platform on the entire PlanetLab testbed (more than 160 nodes at 70 sites) and be able to report on large-scale usage experience.

5 Analysis and Future Work

Based on our experience with the runtime environment abstraction we gained a better understanding of a number of related issues. One of the things that became apparent is the hierarchical structure of such environments. For example, the PlanetLab slice, spanning multiple resources on the testbed, can be modeled as with the same interface for creation and management as the basic RTE. Inside such a distributed RTE (which may be also used to implement a virtual organization) the users would be able to obtain user distributed RTEs. Further, these user-distributed RTEs could implement different levels of sharing: for example, some users may be allowed to execute in them while others may only read data.

Managing the state of a runtime environment is a related issue. For many applications a state suspend/restore functionality would be extremely useful (this would enable migration, for example). Our intuition is that it would be beneficial to abstract this functionality at the runtime environment level. We also currently investigate and comparing different technologies (such as sandboxes and virtual machines) for implementing local runtime environments.

Finally, the emerging WS-Agreement [13] standard could be leveraged for expressing fine-granularity resource allocation to dynamic runtime environments. An interesting related issue is the management of composite runtime environments such environments could be created of multiple remote executions.

6 Conclusions

We have presented OGSI-based abstractions allowing dynamic creation and management for dynamic runtime environments. Our initial experimentation with these abstractions on a large-scale testbed (PlanetLab) shows that they can be usefully applied in practice. We found that OGSI concepts, such as soft-state lifecycle management, SDEs, and service naming (GSH), can be usefully leveraged.

At the same time, we found the set of current standardized services to be lacking. The existence of an authorization service, for example, would allow us to make our work more generic. In addition, the current gridmapfile authorization mechanism is closely tied to UNIX accounts, whereas a more generic RTE identifier (possibly the RTEService GSH, in some canonical form) is needed.

References

1. Butler, R., D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, *Design and Deployment of a National-Scale Authentication Infrastructure.* IEEE Computer, 2000. **33**(12): p. 60-66.
2. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.
3. *dynamic accounts.* http://www.gridpp.ac.uk/gridmapdir/.
4. Kapadia, N.H., R.J. Figueiredo, and J. Fortes. *Enhancing the Scalability and Usability of Computational Grids via Logical User Accounts and Virtual File Systems*. in *10th Heterogeneous Computing Workshop*. 2001. San Francisco, California.
5. Hacker, T. and B. Athey, *A Methodology for Account Management in Grid Computing Environments.* Proceedings of the 2nd International Workshop on Grid Computing, 2001.
6. Chang, F., A. Itzkovitz, and V. Karamcheti. *User-level Resource-constrained Sandboxing*. in *USENIX Windows Systems Symposium*. 2000.
7. *VMware*: http://www.vmware.com/.
8. *User Mode Linux (UML).* http://user-mode-linux.sourceforge.net/.
9. *vserver.* http://www.solucorp.qc.ca/miscprj/s_context.hc.
10. Nahrstedt, K., H. Chu, and S. Narayan. *QoS-aware Resource Management for Distributed Multimedia Applications*. in *Journal on High-Speed Networking, IOS Press*. December 1998.
11. Bavier, A., T. Voigt, M. Wawrzoniak, and L. Peterson, *SILK: Scout Paths in the Linux Kernel*. 2002, Uppsala University.
12. Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. *Xen and the Art of Virtualization*. in *ACM Symposium on Operating Systems Principles (SOSP)*.
13. Czajkowski, K., A. Dan, J. Rofrano, S. Tuecke, and M. Xu, *Agreement-based Grid Service Management (OGSI-Agreement) Version 0.* https://forge.gridforum.org/projects/graap-wg/document/Draft_OGSI-Agreement_Specification/en/1/Draft_OGSI-Agreement_Specification.doc, 2003.