On the Design and Evolution of an Architecture for Federation

Stephen Soltesz, Soner Sevinc, David Eisenstat, Marc Fiuczynski, Larry Peterson
Department of Computer Science
Princeton University
35 Olden Street
Princeton, NJ 08540
<soltesz,ssevinc,deisenst,mef,llp@cs.princeton.edu>

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Distributed Systems; C.2.1 [**Network Architecture and Design**]: Distributed Networks; D.2.11 [**Software Architectures**]: Patterns; C.2.3 [**Network Operations**]: Network Management

General Terms

Federation Architecture

Keywords

federation, virtualization, delegation

1. INTRODUCTION

PlanetLab enjoys a unique position of receiving continual feedback from the needs of an active user community. As a result, there are no clear lines dividing one step from the next in the cycle of design, implementation, and evaluation. Over time, the challenge for PlanetLab maintenance becomes balancing the trade-offs between satisfying short-term needs of user convenience, research agendas, or infrastructure growth, and long-term needs, such as system integrity, sustainability, and coherence, both technologically and for the community. After several years of experience running a global, distributed systems and networking platform, PlanetLab is entering a new design cycle.

The ultimate goal for PlanetLab has always been to enable cooperation among all principals: service users, infrastructure owners, and researchers who invent or maintain services. As the number of independent infrastructures grows, and the number of organizations that hope to participate either in the development of PlanetLab or the maintenance and deployment of PlanetLab software, there is a new tension that spans beyond the original principals to the cooperating organizations who hope to federate.

Drawing on the last several years of experience with PlanetLab, this paper highlights the salient tensions experienced while developing support for delegated management and slice creation, independent machine management and slice registration, and federation between cooperating organizations. Ultimately, we propose a generic architecture suited to cooperation between a separable, management authority (MA) and slice authority (SA) that share a simple set of abstractions, slices and nodes, a common name space, and authorization model. This architecture provides the degree of freedom necessary for federation that is not available in a consolidated implementation. As well, the proposed architecture will permit flexible declaration and enforcement of policies for resource sharing and user access.

Section two provides additional context for this discussion. Section three outlines the original design of PlanetLab and section four provides our experience with it. Finally, section five contrasts the original design with a new proposal.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ROADS Warsaw '07, July 11-12, 2007, Warsaw, Poland
Copyright 2007 ACM 978-1-59593-972-2/07/07 ...$15.00.

2. MOTIVATION

Symbiotic and cooperative agreements between infrastructure providers and service providers are common in a variety of markets[5, 3, 2]. In all of these cases, there is a clear separation between service implementation and resource provider. These correspond to two distinct functions, one that manages and provides access to the physical infrastructure, and the other that provides registration, creation, and access to a service on or over the physical infrastructure. Whether the infrastructure is donated or explicitly purchased is not significant to the underlying relationship.

PlanetLab is also a member of this class of service. Designed on the same principle of separation between resource owner and service developer, it has evolved into a consolidated implementation administered by a single group. Until recently, there were too few organizations with a goal of managing a distributed infrastructure to connect users and resource owners to motivate federation as a design goal. Previously, either the infrastructure was all owned by a single organization, such as campus or business IT departments, or access to independent domains was granted to a single group or person, such as PlanetLab.

PlanetLab wishes to mediate the specific split between infrastructure and services in the architecture. Today, pressures on PlanetLab to federate with other distributed infrastructures is stressing the limits of a consolidated implementation. Assumptions exist regarding namespaces, authorization model and policies needed to enforce trust that are incompatible with federation. As PlanetLab continues to develop during a climate where research infrastructures

are more common, to one where they are ubiquitous or indistinguishable from the commodity Internet today, with this proposal we seek to refine the description of the role and responsibilities of the relevant authorities, the abstractions over which they operate, as well as the namespaces, authorization model and policies needed to preserve the trusted relationship between peers.

3. CURRENT DESIGN

PlanetLab has operated, in one form or another, since 2002. The technologies used and the research agendas of users have changed over the years, but the original need has not changed: to create a platform that connects researchers with remote infrastructure owners to enable distributed systems and networking research around the world.

The following sections describe the current design of PlanetLab. We outline 1) how PlanetLab acts as a trusted party, between two mutually untrusting groups, 2) the boundaries between the logical components of the architecture, and 3) the primary abstractions that allow users to negotiate access to PlanetLab. The outline of PlanetLab's high-level contribution will provide a context to evaluate the new design proposed in section 5.

3.1 Trust

Prior to PlanetLab, a researcher who needed access to machines distributed throughout the network, was limited by the number of personal contacts he had and the administrative overhead of contacting as many remote sites as possible to negotiate a personal account for his work. As the number of researchers grows, while the number of remote sites remains limited due to preferred locations, the overall complexity of this negotiation approaches $O(N^2)$.

By acting as a trusted intermediary, PlanetLab reduces the operational overhead for all researchers performing such a negotiation and all site owners permitting access from foreign users to their network. Because, both researchers and member institutions only interacted with a single intermediary, the overall complexity is reduced to $O(N)$.

Of course, for this to work, all principles involved, user, owner, and PLC, must trust one another, and the incentives for each to trust the other must be self-sustaining. Earlier work defined these trust relationships between principles[7], and both experience and the continued growth of PlanetLab has shown them to be sustainable.

By establishing itself as an authority, the effort needed to contact individual sites was consolidated to one organization. Researchers who wished to benefit from the infrastructure managed by PlanetLab would join as users. Owners who wished to support a common effort for the benefit of the community would leverage the remote administration offered by PlanetLab.

3.2 Authority

The private PlanetLab package, MyPLC, bundles two logically independent functions, the Management Authority (MA) and Slice Authority (SA). Figure 1 represents the PLC database separated by the MA and SA components. As indicated by the direction of the arrows, users express their preferences to the SA (u1), while owners express policy to the MA (o1).

A management authority is responsible for some subset of infrastructure machines: providing operational stability for those components, ensuring the components behave according to acceptable use policies, and executing the resource allocation wishes of the machine owner(o2). The MA maintains an authoritative list of machines, what sites they are associated with, and relevant contact information for maintenance.

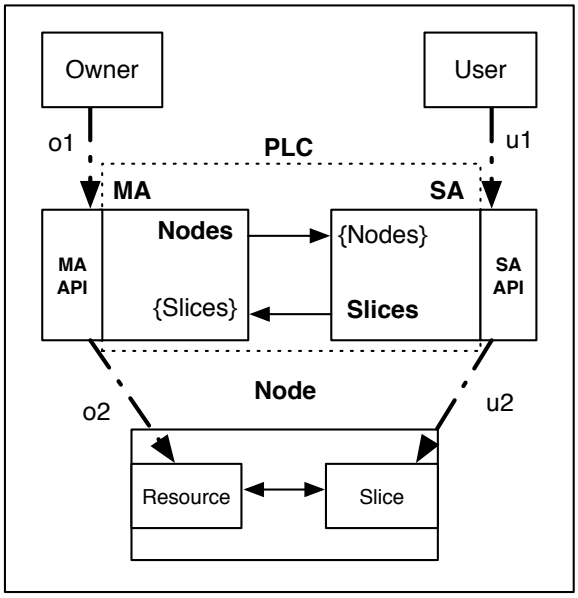


Figure 1: The owner and user interacting through PLC, whose database is divided between the MA and SA, before policy or configuration state reaches the node.

A slice authority is responsible for the behavior of a set of slices, defined next, vouching for the users running experiments in each slice and taking appropriate action should the slice misbehave. The SA maintains a possibly disjoint list of sites and users who can register, create or access slices.

The MA advertises the node list to the SA for assignment to slices. And, in turn, the SA serves as the authoritative source of slices. Working together, the SA advertises the slices to the MA. Ultimately, the machines create slices approved of by the MA and as designated by the user (u2).

3.3 Abstractions: Slices & Nodes

One of the original goals of PlanetLab emphasized the distributed nature of the infrastructure. As such, the primary value to PlanetLab was not access to computing resources, which was already available with clusters, but access to multiple points of presence from machines, also called *nodes*, throughout the global Internet.

The abstraction used to represent a researcher's access to some fraction of all machines in the distributed network is a *slice*. Virtualization makes this possible. The virtual machines that runs on an individual node, is referred to as a *sliver*. Although, many virtualization techniques exist, and any could be suitable for isolating resources on individual machines, PlanetLab has historically used VServer, which is known to perform more favorably under the kind of workloads present on PlanetLab[8].

3.3.1 Slices

From a researcher's perspective, a slice is an infrastructure-

wide network of computing and communication resources capable of running an experiment or a wide-area network service. From an operator's perspective, slices are the primary abstraction for accounting and accountability; respectively, resources are acquired and consumed by slices, and external program behavior is traceable to a slice.

A slice is defined by a set of slivers spanning a set of network components, plus an associated set of users that are allowed to access those slivers for the purpose of running an experiment on the managed infrastructure. That is, a slice has a name, which is bound to a set of users associated with the slice and a (possibly empty) set of slivers.

There are three unique stages in the lifetime of a slice, each corresponding to an action (operation) that can be performed on a slice:

- Register: the slice exists in name only and is bound to a set of users;
- Create: the slice is instantiated on a set of nodes and resources assigned to it;
- Access: the slice is accessed by, and runs code on behalf of, a user.

A slice has to be registered and bound to a set of users before it can be created, and it must be created before being accessed.

Slices are registered in the context of a slice authority, which takes responsibility for the behavior of the slice. A slice is registered only once, but the set of users bound to it can change over time. A slice registration has a finite lifetime; the responsible slice authority must refresh this registration periodically.

Creating a slice effectively configures the slice on a set of nodes; this step can be repeated multiple times. In fact, slice creation often involves two sub-steps: a slice is first instantiated on a set of nodes with only best-effort resources assigned to it, and later provisioned with additional (perhaps guaranteed) resources, for example, for the duration of a single experiment.

An experiment or service then "runs in" a slice. Multiple experiments can be run in a single slice. For each run, the experiment may change parameters but leave the slice configuration unchanged, or it may change either the set of nodes or the resources assigned on those nodes, or both. How rapidly a slice can be reconfigured to support a new experiment depends on the implementation of the instantiation and provisioning operations.

3.3.2 Nodes

PlanetLab nodes are the primary building block of the architecture. For example, a node might correspond to an edge computer or the corresponding PCU used to assist with remote administration.

A node encapsulates a collection of resources, including physical resources (e.g., CPU, memory, disk, bandwidth) logical resources (e.g., file descriptors, port numbers), and synthetic resources (e.g., packet forwarding fast paths). These resources can be contained in a single physical device or distributed across a set of devices (e.g., multiple NICs for instance), depending on the nature of the hardware. However, a given resource can belong to at most one node.

Each node is controlled via a node manager (NM), which exports a well-defined, remotely accessible interface. The node manager defines the operations available to user-level services to manage the allocation of node resources to different users and their experiments.

A management authority (representing the wishes of the owner) establishes policies about how the node's resources are assigned to users.

It must be possible to multiplex (slice) node resources among multiple users. This can be done by a combination of virtualizing the node (where each user acquires a virtual copy of the node's resources), or by partitioning the node into distinct resource sets (where each user acquires a distinct partition of the node's resources). In both cases, we say the user is granted a sliver of the node. Each node must include hardware or software mechanisms that isolate slivers from each other, making it appropriate to view a sliver as a "resource container."

4. EXPERIENCE WITH THIS DESIGN

PlanetLab has anticipated the federation of infrastructures[7]. Yet, the currently centralized implementation introduces a tension on the interaction between peers in terms of code development, deployment, management, resource naming, and operational scalability.

The following section illustrates by example a variety of issues that we have needed to solve in recent times. The sections that follow, outline how we are approaching a generalized architecture for federation from the starting point of a centralized implementation.

4.1 A Brief History

PlanetLab began with a distributed design. The expectation was that PlanetLab's architecture would reflect the design of the distributed systems it hoped to enable. PlanetLab provided a minimal Boot Manager that would boot strap the first order services on the machine before it joined the rest of the infrastructure.

However, this approach proved vulnerable to consistency errors between PlanetLab Central (PLC) and state preserved on the node. As a result, the implementation and functionality separated across the node and PLC was consolidated. Now, the node serves as an explicit cache for state maintained at PLC.

Concordant with this shift, PlanetLab developed MyPLC, a software package complete with all the backend services needed to run the production PLC. In turn, this enabled others to establish and manage their own distributed infrastructure with which PLC could federate[7].

Recently, PlanetLab has agreed to federate infrastructures with OneLab Europe[1]. Prior to this agreement, PLC was the only management authority responsible for developing software and maintaining machines that were compatible with the PlanetLab implementation. Now, OneLab will act as a MA and maintain an independent set of machines. For the user though, there will be no difference between machines managed by OneLab and machines managed by PlanetLab. A user will register, create, and access slices on these machines *as if* they were managed by PLC, and vice versa.

As PlanetLab continues to grow in geographic scope, we expect other autonomous regions to act as independent MAs like OneLab. However, the result is a tension between the consolidated implementation of PLC and the distributed requirements of independent management authorities that wish to maintain a common user experience. As well, many

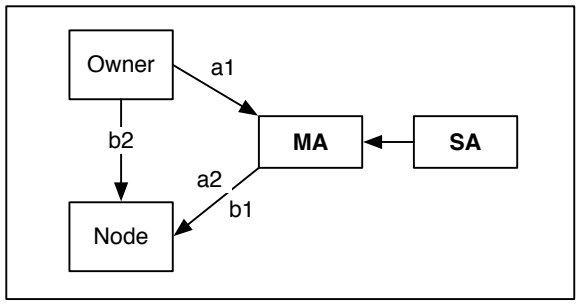


Figure 2: Owner delegated management expressing policies to the MA or the node.

other distributed platforms already exist, such as VINI[4], Emulab[9], and Everlab[6], among others. These platforms often provide access to novel resources not available on others, further motivating the need to access resources distributed across management authorities.

4.2 Delegated Management

Owners delegate administration of local machines to PlanetLab. This leverages the work done by PlanetLab to vet users, develop code, provide technical and abuse support, as well as issue tracking over time. There is nothing that prevents site administrators from performing these steps themselves. However, there is clearly an advantage to delegating to a trusted party.

The mechanism requires a minimal configuration:

- Remote power control with registered network address.
- Node hardware and registered network address.
- A boot image with a shared secret known only to the node and PLC.

The remote power control is engaged to power on the node. The node in turn boots from the boot image and the shared secret is used in an authentication hand-shake with PLC. Finally, the Boot Manager retrieves a base image for the node and installs or updates the system software on the persistent disk. Clearly, as described the network administration is outside the scope of PlanetLab. It is the responsibility of local administrators to ensure that machines delegated to PlanetLab have open network connectivity.

This does not imply loss of control for the local administrator. PlanetLab has always maintained a node owners ability to express preferences for how resources are allocated on his machines and monitor the traffic entering the network. In the original PlanetLab implementation when the authoritative state maintained by PlanetLab was distributed across both PLC and the nodes, node owner preferences were recorded on the node. Figure 2 illustrates the alternative paths to the node from either the MA of PLC or the node owner. Because the node acted as an authoritative repository for resource allocation policy, it was possible for a mismatch between the owner's preferences (b2), the user's expectations, the record maintained by PLC (b1), and what NM ultimately enforced. The solution we adopted was to consolidate all owner policy at PLC. Now the owner registered his preferences with the MA (a1), which was later propagated to the node (a2).

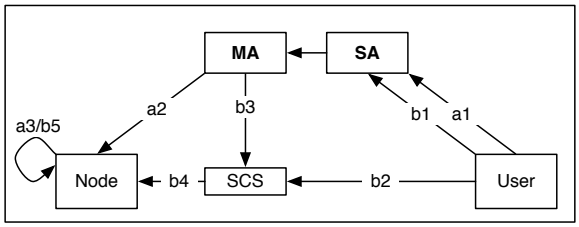


Figure 3: Delegated slice creation using the default asynchronous method of slice delivery to the node, versus an indirect SCS.

While consolidating policy at PLC resolved the dilemma for managing a single infrastructure, when multiple management authorities exist, as they do with federation, the issue returns. Two independent organizations again introduces multiple sources of resource preference and policy information. Clearly, though, whether policy decisions are enforced globally, at the authority level, or locally at the node, the path from root authority to the node should be unique.

4.3 Delegated Slice Creation

Users delegate the operation of creating accounts to PLC. This leverages the work done by PlanetLab to manage machines, solicit participation from machine owners, and provide maintenance for the system software. Of course, a user need not join PlanetLab for access to a remote account, but there are clear advantages to doing so.

A consolidated implementation implies that all state on the node arrives one way or another via the approval of the central authority. This is no different for slice creation. Node Manager periodically polls PLC for a list of all the slices that should exist on this node. If the slice does not exist, it is created. If the slice exists when it should not, perhaps due to an expiration, it is deleted.

However, because of the polling, after a user registers a slice at PLC, a delay is introduced between the time of registration and the time it is created on the node and available for access by the user. This delay is unacceptable for certain experimental setups, such as Emulab. The Emulab project uses Planetlab slices as a resource for its user's experiments. For this, Emulab requires synchronous slice creation from the time their user submits an experiment to the time it runs. However, PlanetLab only offers a default, asynchronous slice creation service (SCS). To address this conflict, Emulab has implemented an alternative SCS that operates on behalf of a PlanetLab user. In doing so, the slice (at registration time) *delegates* its creation to Emulab's SCS, rather than the default SCS.

Because, PLC serves as the slice authority (SA) there are no other valid sources of slice names in the system. However, delegation of slice creation offers additional flexibility in how a registered slice is delivered to a node for creation. A user with an account at both PlanetLab and Emulab can delegate Emulab as the principal to create their service on specific nodes.

To illustrate these two scenarios, figure 3 shows two paths that the information needed by the Node Manager could take. Along the top path, a user registers a slice name with PLC's SA (a1), later this information is propagated to the NM (a2), and created (a3).

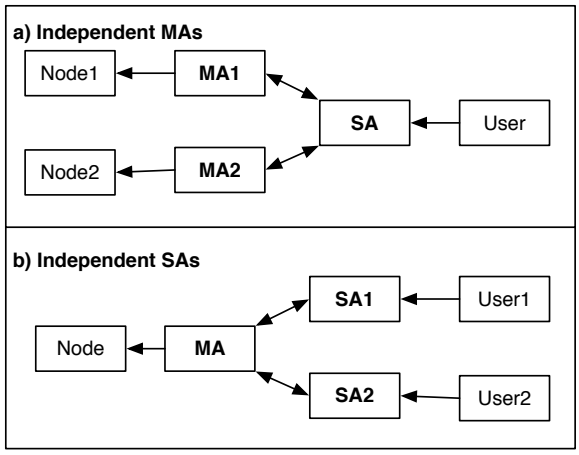


Figure 4: Federation using independent authorities for management or slice registration.

The second path of figure 3, also begins by registration at PLC's SA (b1), followed by the user handing control over to a SCS (b2). Next, the SCS issues a GetTicket() operation (b3), which is a cryptographically signed and serialized representation of the intention of the MA to create a slice on the given node. The system is semantically identical. The only difference is that state previously within PLC is externalized. Finally, the SCS delivers the signed ticket to the NM (b4) and the slice is created (b5).

As with delegated management, a consolidated implementation allows for delegated slice creation across users within PlanetLab. However, with federation, it is not clear whether delegated slice-creation can or should exist across federating peers. For instance, can an authority issue tickets for machines managed by another MA? Should the user interact directly with the second MA? Or is it necessary for the second SCS to have an independent relationship with all MAs it wishes to support?

4.4 Independent Authority

We are currently in the process of federating with OneLab Europe. This agreement will have traits common to many of the federating agreements to come. In this particular case, each will maintain the reciprocal agreement to host the other's slices on nodes they manage. OneLab will take responsibility for the machines within their geographic region and provide all aspects of infrastructure management for these machine: software development and deployment, a node registry, periodic monitoring and maintenance, as well as the first and last point of contact for dispute resolution between actions on machines they manage and conflicting expectations of third parties by auditing user activity to maintain a "chain of responsibility" from the slice authority to the responsible user[7]. As well, each will provide user accounts to researchers for the creation of services, a slice registry, and assign roles to individual users for specific operations within the system. These responsibilities correspond directly to the MA and SA.

Because the only implementation for federation between peers is currently based on MyPLC, both partners have an MA and SA. This leads to a situation where there are *two infrastructures* and *two namespaces*, respectively. With these two components, the following descriptions outline two possible configurations using the current components for federation. Extracting from the limitations of these, section 5 presents the general architecture.

4.4.1 *Management: Two Infrastructures, One namespace*

The original plan between OneLab and PlanetLab was to federate only the management authorities. This would create a model where nodes that were delegated for management to one organization were delegated-by-proxy to the second organization, since the owner still delegates directly with only one MA. For slice users these nodes would operate with the same API and behave similarly. As well, because there is only a single SA, there would be no confusion about user roles or the issuing and honoring of slice tickets. Figure 4a illustrates independent management authorities with a single SA.

With such an approach, a single slice registry interacts with both management authorities, thereby allowing users who join the single registry to access machines operated by both MAs. With the current implementation, this approach would be limited to a centralized registry. However, OneLab Europe is an independent organization with unique research agendas and methods for vetting and allowing users to join their infrastructure. A centralized registry would not permit the autonomy they require.

4.4.2 *Slice Creation: One Infrastructure, Two namespaces*

Because OneLab based their infrastructure software on MyPLC, they inherit a fully functional SA. Leveraging this component as part of the federation agreement, each group has the ability to register and create slices independently of each other. While this approach would distinguish roles between the two SAs, it complicates the adoption and integration of novel features within the infrastructure. Figure 4b illustrates independent slice authorities peering with a single MA.

As distributed network infrastructures become more common and users more aware of them, users will expect either (a) their favorite platform to integrate the popular features of the others, or (b) the ability to seamlessly access the features of any platform from a single account. Clearly, the first case is not tenable in all situations, and in the second case care is needed to preserve compatibility without sacrificing flexibility or control of the platform development.

5. DESIGN EVOLUTION FOR FEDERATION

The preceding described the motivation for and the limitations of a consolidated design and implementation. The following discussion describes which aspects of the architecture need to be explicit to allow the implementation to be de-consolidated to better facilitate cooperation in a federated environment. The principle users, owners, and two primary authorities remain and are not discussed further below. As well, the primary abstractions used between authorities remain slices and nodes. However, the normative interface is expanded and compatibility with the architecture is extended to include globally unique names and identifiers for the mapping of resources and slices. As well, the authorization model is refined such that principles are able to access resources through a federated system and express

policy preferences to the relevant authority.

5.1 Selective Consolidation

The number of independent authorities increased with the split between MA and SA. Now in a federated architecture, the number increases again. Fortunately, the tasks performed by the components does not change. Therefore, the challenge for this architecture becomes specifying these tasks sufficiently to allow conditional separability, thereby allowing either a single or an arbitrary number of independent authorities to implement each task from first dependency to ultimate action.

Using the management authority as an example, the sequence of tasks roughly follows: code development, deployment, authority operation, node installation, slice creation. A similar path could be traced for the slice authority. While code development occurs at the beginning and slice creation occurs at the very end, it should be possible to federate using any combination of consolidated or separated components.

As an example, there are two forms of federation using slices. In the first all slices registered by the SA will be exposed to the MA, and created on the node. Alternately, the MA could grant only a single slice, leaving the SA to subdivide their single slice into an arbitrary number of subslices. In the second case, the separation occurs closer to the end of the list of tasks and on the node, as opposed to at the authority level.

As well, the federation with OneLab has included several stages where code sets were forked and then merged again. However, once the implementation separation approaches all components including the code that implements the normative interfaces, the remaining aspect that must be preserved is the abstract interface.

5.2 Trust

The trust previously consolidated within PLC is now distributed across four components, the SA and MA of both organizations. The trust between owner and MA as well as the user and SA does not change, since from their vantage, they only interact with a single entity both originally and in a federated environment. The only weakness that could exist would be between the interaction of MA and SA.

However, the trust between MA and SA existed previously within PLC, all be it implicitly. The difference in the architecture is that the state exchange between MA and SA is explicit and external. It then follows that the degrees of trust that was previously implied is now explicit between the MA and SA. As well, now the trust may be across multiple peers. But the relationship between any pair is not fundamentally different from its original relationship in the consolidated implementation.

5.3 Names and Identifiers

The previously consolidated namespace and identifiers within PLC is now distributed across components, namely the SA and MA of both organizations. In order for each component to uniquely address objects in the system and across authorities, externalized names and identifiers are needed. So, although the state is distributed, the architecture preserves one namespace.

For this end, the federation architecture will define unambiguous identifiers, simply called Global Identifiers (GID), for the set of objects that make up the federated system. GIDs form the basis for a correct and secure system, such that an entity that possesses a GID is able to confirm that the GID was issued in accordance with the SFA and has not been forged, and to authenticate that the object claiming to represent the GID is the one to which the GID was actually issued.

A name registry maps strings to GIDs, as well as to other domain-specific information about the corresponding object (e.g., the URI at which the object's manager can be reached, an IP or hardware address for the machine on which the object is implemented, the name and postal address of the organization that hosts the object, and so on).

Following from the separation of SA and MA, there are two registries, a node registry and a slice registry, both of which define a hierarchical name space corresponding to the hierarchy of authorities that have delegated the right to create and name node and slice objects, respectively. These registries assume a top-level naming authority trusted by all entities, resulting in names of the form:

$$toplevel_authority.sub_authority_i.name$$

For example, "onelab" and "planetlab" might be top-level authorities; it is possible that other similar authorities would federate using this model.

This is not to imply that all federation is strictly among top-level authorities, since even in the context of a single top-level authority, we allow for multiple autonomous MAs that agree to federate their resources. As well, to maintain the trust relationship, no sub-authority would join an untrusted top-level authority, or does so at the risk of their own credibility.

The node registry maintains information about a hierarchy of management authorities, along with the set of nodes for which the MAs are responsible. This registry binds a human-readable name for nodes and MAs to a GID, along with a record of information that includes the URI at which the node's manager can be accessed; other attributes and identifiers that might commonly be associated with a node (e.g., hardware addresses, IP addresses, DNS names); and in the case of an MA, contact information for the organization and operators responsible for the set of nodes. For example,

$$planetlab.us.princeton.machine1$$

might name a node at the Princeton site of PlanetLab's US sub-authority. In this case, the planetlab.us.princeton management authority is responsible for the operational stability of the set of nodes on their network.

The slice registry maintains information about a hierarchy of slice authorities, along with the set of slices for which the SAs have taken responsibility. This registry binds a human-readable name for slices and SAs to a GID, along with a record of information that includes email addresses, contact information, and public keys for the set of users associated with the slice; and in the case of an SA, contact information for the organization and people responsible for the set of slices. For example,

$$planetlab.fr.inria.dali$$

might name a slice created by the PlanetLab SA, which has delegated to the French SA and then to INRIA, the right to approve slices for individual projects (experiments), such as

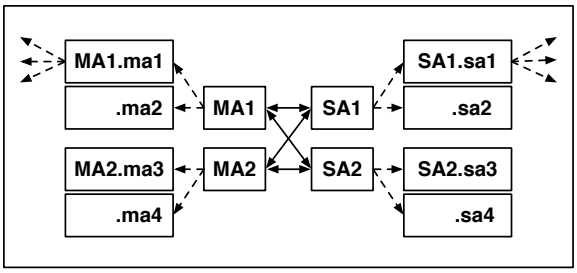


Figure 5: A general view of peering and hierarchical registry of MAs and SAs.

Dali. PlanetLab defines a set of expectations for all slices it approves, and directly or indirectly vets the users assigned to those slices.

Note that both the OneLab and PlanetLab management authorities are expected to maintain an operational set of nodes capable of hosting experiments, and their respective slice authorities are expected to support slice creation on behalf of network and distributed systems researchers. Because it is possible that other related facilities will federate with OneLab and PlanetLab, and there will be other uses of the greater federated system, we allow for the possibility that other top-level slice authorities may support other policies and purposes. For example, there could exist a top-level slice authority that permits slices running for-profit services.

Also note that the registries may be distributed, where a server that implements one portion of the hierarchy includes a pointer (URI) to a server that implements a sub-tree of the hierarchy. We expect slice and management authorities will often implement a registry server for the sub-tree of the hierarchy for which they are responsible. Figure 5 illustrates the generalized relationships discussed so far.

5.4 Authorization

Previously, there was no hierarchy of slice authorities. And, the roles assigned to users existed in a flat role-space. With the introduction of a hierarchical registry, the roles for organizations, sites, or users can become a function of the position within the graph. And it is possible to allow the permission to perform an operation to extend down the graph. As the number of authorities are increased each parent authority grants a subset of permission to children.

Two primary classes of resources are protected by the authorization mechanisms of this model. The first of these is slices, the containers for user-level experiments. The second is the collection of physical and logical resources implemented as, or made available by, nodes.

For example, a SA (on behalf of a user) registers a slice name, binds it to a GID, and thus produces a slice. Thus, the authorization model for slice creation is based on controlling access to the slice name space. And, in particular, the position within the hierarchy that the name is positioned. The slice name space consists of a naming hierarchy rooted at a top-level SA. Below the top-level SA, intermediate nodes in the naming tree represent intermediate slice authorities, while the leaf nodes represent the slices themselves. Before registering a sub-node of its portion of the name space, a slice authority must use an off-line process to verify that it is willing to vouch for the sub-authority, slice, or user. It is assumed that each SA has been delegated authorization authority over its portion of the slice name space, and need not refer registration requests to a higher level authority. For fail-safe reasons, a slice registration has a finite lifetime, and it is necessary to periodically refresh the registration.

On the other hand, physical resources are encapsulated as nodes. Each node has an associated management authority, which represents the node's owner. The MA is responsible for defining and implementing the authorization policy governing use of the node. Tickets are used to implement this model, where a ticket represents a principle's right to (1) create a sliver (perhaps with some initial resources bound to it) on a node, (2) bind node resources to an existing slice, and (3) control a running slice. All three rights can be delegated. A ticket is essentially a resource specification signed by a node's MA, granting rights to allocate resources on one or more nodes. The holder of the ticket must go back to the node to split it into two tickets but the expectation is that other tickets can be delegated by the slice identified in the ticket.

5.5 Policy

Implied but missing is an explicit discussion of the policy used by both authorities and how conflicts are expressed and resolved. Policies will be applied at three distinct levels: 1) between federating peers, 2) access control for individual components, i.e. for slice creation, and 3) the resource allocation on a given node.

Policies will be expressed by all principles involved: owners, users, peers, and authority administrators. Between federating peers, there will need to be a mechanism for mediating and resolving the overlap of policies held by one relative to the policies enforced by the other. Because tasks are separable and the objects have role relationships defined by position in the registration hierarchy, many policy statements will become descriptions of operations and whether a particular position in the graph can perform the given operation at every position below the graph. One very sensitive operation is administrative access to nodes. This operation might be reserved only for a root MA, as it is with PlanetLab. Child MAs that may share the root as a parent would not be given this capability as a matter of policy.

At the second level, policies would control access to the creation of slices. As discussed earlier, how users are allowed to join and whether or not they are granted permission to register, create, or access slices follows from their role today. In terms of policy, there are two extremes of using slices to federate: all unique slice names are shared with the peering MA, or a single slice is exposed to the peering MA, and the SCS of the SA has privilege to subdivide this slice into smaller slices. The domain of the first option applies at the authority level, while the second applies to the nodes themselves. This leads to the final case, where the resource allocation occurs on a given node.

Node allocation policy would come in the form of resource limits for total disk space, bandwidth limits, and number of slivers on a node. For instance, Emulab is limited by the total number of slices they can create (10), due to the current policies of the PlanetLab SA. But, rather than the number of total slices, an alternate policy would count the total number of slivers assigned across all machines.

6. CONCLUSION

This progression of steps has taken the reverse order or-

dinarily associated with system design. The traditional approach would begin with object specification, interface declarations, user roles, and access controls for these objects. The evolution of PlanetLab has followed the reverse path, at times explicitly avoiding the definition of hard edges around interfaces and objects to avoid the pitfall of balkanization that can occur over time for missed design points.

As cooperation between organizations becomes common, the next phase of the design cycle will be refining the interfaces for the components described here. The degree of success PlanetLab has had to this point, reflects the suitability of this alternative approach when there are too many factors to consider at the beginning. Where rather than designing first, the design evolves as a function of the mutually beneficial resolution of the natural tension that comes from the competing interests of cooperating groups.

7. REFERENCES

[1] Onelab project, 2007. `http://www.one-lab.org`.

[2] J. Altmann, M. Ion, and A. A. B. Mohammed. Taxonomy of grid business models. In *Lecture Notes in Computer Science*, volume 4685, pages 29–43, 2007.

[3] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[4] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: Realistic and controlled network experimentation. In *Proceedings of ACM SIGCOMM 2006*, Pisa, Italy, September 2006.

[5] N. Feamster, L. Gao, and J. Rexford. How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, 2007.

[6] E. Jaffe, D. Bickson, and S. Kirkpatrick. Everlab - a production platform for research in network experimentation and computation. In *Proceedings of the 21st Usenix LISA Conference.*, Dallas, Texas, Nov 2007.

[7] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building planetlab. In *Proceedings of the 7th USENIX Symposium on Operating System Design and Implementation (OSDI '06)*, Seattle, WA, November 2006.

[8] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *EuroSys 2007*, Lisbon, Portugal, March 2007.

[9] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. 5th OSDI*, pages 255–270, Boston, MA, Dec 2002.