Implementing SFA Support on an Established HPC-flavored Testbed: Lessons Learned

Luke Bertot, Lucas Nussbaum, David Margery

Implementing SFA Support on an Established HPC-flavored Testbed: Lessons Learned

Luke Bertot*, Lucas Nussbaum*, and David Margery†
*Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
†Inria, F-35000 Rennes, France
firstname.lastname@inria.fr

***Abstract*—The Slice-based Federation Architecture (SFA) is the de facto standard framework for managing testbeds, federations of testbeds, and users access to these federations. It is the foundation of most major testbed federations in the world, including GENI and Fed4FIRE. However, there remain some testbeds that were designed and grew outside of this world, making different, interesting and sometimes better design choices. In this paper, we describe how we added support for the GENI Aggregate Manager to the Grid'5000 testbed, a major testbed focused on HPC and Cloud that was developed for the most part independently for the last 15 years. From this experience, we draw some lessons and recommendations that could help improve the testbed management ecosystem.**

***Index Terms*—testbeds; testbed management framework; Slice-Based Federation Architecture (SFA); resource selection, reservations, configuration**

I. Introduction

As the internet gained widespread adoption in the late 1990s, new types of distributed systems, such as peer-to-peer networks and content distribution networks, began to emerge. For computer scientists, studying and experimenting was a complicated process, which often required borrowing access to machines in other institutions and dealing with heterogeneous configurations and software. Faced with these difficulties, the community moved to organize shared testbeds.

These testbeds would be centrally managed and offer a more consistent method to access resources over multiple sites, on which experiment isolation would be achieved using techniques such as network slicing, virtualization, or bare metal loading. The major examples of such community testbeds were Emulab [1] focused on network experimentations using emulation, PlanetLab [2] focused on highly distributed applications, ORBIT [3] for wireless networks, and DETER [4] focused on security. These testbeds quickly became an essential part of the study of distributed systems.

Recognizing the importance of such testbeds, the US National Science Foundation (NSF) contributed to setting up the Global Environment for Networking Innovation (GENI) Project Office to develop the next generation of testbeds. By setting it up as a federation of testbeds, GENI [5], [6] would be able to provide a wide array of different resources over a large number of physical locations through a common interface.

The work on this project has been supported by the Horizon 2020 Fed4FIRE+ project, Grant Agreement No. 723638.

In the GENI architecture, three parties interact: the user, the federation, and the testbeds. The federation provides user authentication and authorization. The federation *Clearinghouse* issues user certificates and signed credentials which authorize users to interact with federation-wide namespaces called *slices*. Using these certificates and credentials, users can contact the different testbeds through an aggregate manager (AM) to add resources to their slice. Resources are provided as part of testbed-wide namespaces called *slivers* which are added to the user slice. This setup delegates user and experiment management to the federation, letting AMs decide which federation's credentials to accept. GENI proposes a set of standards for the different elements of this *Slice-based Federation Architecture* (SFA). The current version of the AM API is AMv3.

Today, GENI APIs are widely seen as the de facto standard for testbed management and are used by other testbed federations such as Fed4FIRE in Europe. However, other testbeds have been designed and developed outside of the SFA world. In this paper, we discuss the challenges we faced, and the lessons we learned, implementing a GENI AM in an established testbed: Grid'5000. We hope that this discussion will be useful to testbed designers and operators, to understand what implementing an AM encompasses. We also hope that it will be useful to identify future SFA evolutions.

The paper is structured as follows. Section II provides some background on the Grid'5000 testbed. Section III offers a comparison of GENI and Grid'5000 APIs and presents Grid'5000's implementation of an Aggregate Manager. In section IV we focus on how the differences between GENI and Grid'5000 affect our work, and how we would hope GENI and Grid'5000 could change as a result. Finally, we discuss other testbed control frameworks in section V before concluding.

II. Background on Grid'5000

The Grid'5000 project [7] was initiated in 2003 by the French HPC (High Performance Computing) research community. At the time, the focus of this research community was on Grid computing, and there was a clear need for a large scale highly reconfigurable testbed in order to experiment at scale. Many of Grid'5000's design decisions are inherited from that time and inspired from the HPC world, for example like most nationwide computing grids, it is distributed over several sites, and it uses an HPC resource manager (batch scheduler) to manage resources.

Over the years, the focus of the testbed has evolved, but remained rooted in the French HPC and distributed systems community. Its focus today is similar to testbeds such as Chameleon [8] or CloudLab [9], covering topics like HPC, AI, Clouds, Edge Computing, Big Data. The services it offers are similar to those offered by the aforementioned testbeds [10]: bare metal provisioning, network isolation, monitoring services, etc. It also offers similar hardware: x86 and ARM servers, HPC networks (e.g. InfiniBand, Omni-Path), GPUs, etc.

Since 2017 in the context of the EU H2020 Fed4FIRE+ project, Grid'5000 joined the Fed4FIRE testbed federation, had to support SFA, and thus implement and operate an AM. This raised a number of challenges, detailed in the next section.

III. Implementing the Slice-based Federation Architecture in Grid'5000

A. Technological Overview

1) GENI AMv3 API: This API is based on XML-RPC over HTTPS. Calls are made using an SSL client certificate. This user certificate is issued by the federation and is used by the AMs to filter access to resources. Additionally, for most operations, users will provide a *credential* in the call parameters (an XML snippet signed by the federation *Clearinghouse*). The credential describes the permissions given to the subject of the client certificate (the user) over the target slice. Using the API, users are able to add AM-local *slivers*, representing testbed resources, to their federation-wide *slice*.

In a typical use case, the user would first query the AM for its capabilities and available resources, presented in a *advertisement RSpec*, using the **getVersion** and **listResources** API calls respectively. To claim resources for a slice, the user sends a *request RSpec* in a **allocate** call to every involved AM. The AMs book the requested resources in corresponding slivers and answers the **allocate** call with a list of slivers and a *manifest RSpec* describing the booked resources. The user must then confirm their booking using a **provision** and a **performOperationalAction** call, during which the AMs will start and configure the booked resources. At any time, the user can use **status** to find the logical and operational state of the chosen slivers, and **describe** to obtain a manifest of chosen slivers. Once an AM shows a provisioned sliver in the *ready* state, the user can connect to the underlying resources to perform their experiment. Once finished, the **delete** call is used to release the resources booked in the chosen sliver.

RSpecs are XML snippets used by the AM API to describe resources, and come in three flavors: *advertisement*, *request*, and *manifest*. RSpecs contain `<node>` and `<link>` elements, representing respectively computational resources and network links. Nodes are advertised with a `component_manager_id` (the testbed in charge of the node), a unique identifier for the node (`component_id`), its type (bare metal, virtualized, ...), disk images available, and possibly hardware types and network interfaces. In a *request RSpec*, the user must only specify a label, the `client_id`, the component manager handled by the AM where the node will be located using the `component_manager_id`, and the node type. Additionally users can specify an image to deploy, a hardware type or a specific node. *Manifest RSpecs* mostly copy the content of the corresponding *request RSpec* completing it with the identifiers of the allocated resources and additional login information.

2) Grid'5000: For most use cases, Grid'5000 users do not access the testbed directly through the API. Users usually connect via SSH to the sites' front-ends and use available commands to manage resources. Grid'5000's testbed control infrastructure is based on OAR [11] (an HPC resource manager) and related tools. Each Grid'5000 site runs an independent instance of OAR. Users looking for specific resources are invited to use the Grid'5000 website[1] to find the clusters (sets of homogeneous nodes) available at different sites.

The granularity of resource bookings can go from a whole cluster to a single CPU-core. Sub-node level bookings rely on Linux *cgroup*/*cpuset* mechanisms to maintain isolation between jobs. OAR also allows for the building of complex requests based on resource hierarchies. A user can choose to book ten CPU cores using a `core=10` request or can request specific topologies such as having one core on two separate CPUs of five different nodes all belonging to a single cluster using `cluster=1/host=5/cpu=2/core=1`. These hierarchies can further be constrained through property filtering, to restrict execution on specific cluster nodes, or to filter resources based on intrinsic properties such as memory size or network bandwidth. Property filtering relies on an SQL request allowing users to write complex constraints. A single booking can contain multiple resource hierarchies each with their own property constraints. Additionally users should specify a job walltime (duration), optionally a command to execute, and can specify a job type. Job types can be used to request the default system image (avoiding bare metal deployment) or the ability to deploy disk images, which is performed using a second tool, KaDeploy [12]. Resources are automatically released after the walltime is reached, or if the provided command finishes, or if the job is forcefully terminated using OAR.

Grid'5000 offers a REST API, grouping the different services. Part of this API, called the *reference API*, offers a functionality similar to SFA's **listResource** by providing information on available resources, using HTTP GET on `/sites/<site>/clusters/<cluster>/nodes/<node>`. The *jobs API* allows users to create and manage OAR jobs, fulfilling the function of the AMv3 **allocate** and **provision** calls. Jobs are created using HTTP POST requests specifying walltimes, commands, resource hierarchies, properties and types. Following the REST principles, these operations create a new endpoint for the jobs that users can query to track the state of the OAR job or DELETE to end the job. Conversely the *deployments API* endpoints can create and monitor KaDeploy's installation of disk images, fulfilling part of the **provision** call. The API also offers endpoints to alter

[1] https://www.grid5000.fr/w/Hardware

TABLE I
GRID'5000 API CALLS TRIGGERED BY AMV3 API CALLS

getVersion	No Grid'5000 API calls
listResources	GET `/sites/<site>/status` GET `/sites/<site>/clusters/<cluster>/nodes/<node>` GET `/sites/<site>/internal/kadeployapi/environments`
allocate*	POST `/sites/<site>/jobs`
provision*	POST `/sites/<site>/deployments/`
status & describe*	GET `/sites/<site>/jobs/<job_id>` GET `/sites/<site>/deployments/<id>`
delete*	DELETE `/sites/<site>/jobs/<job_id>`
performOperationalAction	No Grid'5000 API calls
Renew	Not currently implemented

*: these calls require user impersonation. Users are found/created using: GET `/users/engines/fed4fire/external_id`

network topology, monitor power usage, and access account management.

B. Design and Implementation of an AM for Grid'5000

The *geni-tools* suite[2] provides a base implementation of an aggregate manager, providing parsing XML-RPC requests, structuring responses, and management of certificates, credentials, slices, and slivers. Table I shows what calls are made to the Grid'5000 API for a specific GENI AM call. The AM is authenticated with an SSL certificate allowing it to impersonate Grid'5000 users during calls.

1) Access Management: The AM is configured to accept any calls passed with a valid certificate signed by the Fed4FIRE federation. However most of our internal tooling relies on the existence of local user accounts. We therefore had to implement the dynamic creation of Grid'5000 user accounts for federation users, and also the mapping of existing Grid'5000 user accounts to federation user accounts for requests coming through the AM.

Our user management service (UMS) has been updated to add the possibility for accounts to have external identifiers, allowing us to add a *Fed4FIRE user URN* (Uniform Resource Name) to accounts. The UMS was also extended to provide a method for retrieving an account associated with an URN and, if none exists, creating new a account on the fly using the information provided in the user certificate. Accounts created only have a username, the email address, and an SSH public key, all taken from the certificate key. Such accounts are only valid for a month, and extending the validity requires users to provide information about their identity and their affiliation. With this setup, new users coming from Fed4FIRE can immediately use the testbed, while giving us visibility on more long-term users.

2) Listing Resources: The AM builds the *advertisement RSpec* by interrogating the *reference API*. Since this information is public (as it is available on the Grid'5000 wiki), this call only checks for a valid user certificate and does not check whether the caller has a valid Grid'5000 account.

The AM advertises every node in the testbed. The site to which a node belongs is exposed as part of the *component_manager_id*, the cluster is exposed as the *hardware_type*, and the node URN is built from the corresponding *reference API* path. OAR properties other than the cluster are not exposed through the *advertisement RSpec*. For each node, the list of current Grid'5000 managed disk images is provided.

3) Allocation and Provisioning: When faced with an **allocate** request, the AM will first check the validity of the user certificate and the credentials provided by the federation, and find the corresponding Grid'5000 user account or create one. The *request RSpec* is then parsed for relevant node requests. These nodes are then started and added to the user slice. Lastly, if OAR manages to book all the requested resources for immediate use, the corresponding manifest is generated and sent back to the user, otherwise the allocation fails.

During allocation, each node requested from Grid'5000 is started using a single-node OAR job. These jobs are then turned into slivers added to the user slice. Nodes requesting a specific disk image will be booked using the OAR `deploy` type, whereas other nodes will use the `allow_classic_ssh` type (providing the default system environment, without bare metal deployment). The walltime is set to the requested end date or the end of the user credential validity. Internally, the AM sets the expiry of the slivers at 10 minutes.

During a **provision** call, the AM extends the validity of the provisioned slivers to the end of the walltime of the OAR jobs, and deploys requested disk images to the provisioned nodes.

4) Node Access: Once their nodes reach the `geni_ready` state, users will be able to connect to them using SSH. Nodes started without a specific disk image allow users to connect to their personal account on the node and access their NFS home directory. On nodes started with a specific disk image, users will connect to the root account. In all cases users will need to first connect to one of the Grid'5000 SSH access gateways, and the connection will use their Fed4FIRE user certificate's private key as identity. The information on how to connect to the gateway and to the node is available in the *manifest RSpec* produced by the **allocate**, **provision**, and **describe** API calls.

5) Deletion: If no provisioning has been performed in the ten minutes following an allocation, the allocated slivers are marked for expiry. Expired slivers are deleted by the AM, stopping the underlying OAR job and freeing resources. Additionally resources can be freed using the **delete** API call.

C. Current Status

After the implementation of all of the above, it is now possible to reserve and provision Grid'5000 resources through Fed4FIRE's standard GUI tool (jFed[3]). Additionally, support for *Experiment Specification (ESpec)*, a Fed4FIRE-built solution for experiment packaging, also works immediately. The AM necessitated 2154 new lines of Python code on top of

[2]https://github.com/GENI-NSF/geni-tools

[3]https://jfed.ilabt.imec.be/

the code provided by *geni-tools*, and makes use of an external 1215 lines from the *python-grid5000* library.

IV. Divergence between GENI and Grid'5000

In this section we discuss the points of contention encountered while implementing GENI AM on Grid'5000, and how they relate to the difference in objectives between GENI and Grid'5000. Where relevant, we also make recommendations.

A. Support for Multi-Site Testbeds

Grid'5000 is a single testbed distributed on multiple sites. Every site is managed by the same technical team and user accounts are centrally managed, but resource bookings happen on a site-by-site basis. The GENI AM API is designed to allow AMs to aggregate multiple testbeds, each identified by their `component_manager_id`. Since most testbeds run their own AM, tools such as jFed have poor support of AM with multiple component managers. However giving each site its own AM would turn internal links within Grid'5000 into AM links. Using a single Grid'5000 AM is thus more consistent with Grid'5000's internal structure.

We have therefore chosen to stay with the initial design and expose each Grid'5000 site as a component manager of the global Grid'5000 AM. Poor support for that design is mitigated by the fact that site information is only necessary when users do not request a specific node or cluster. Our implementation overrides component manager information to always match the requested resource. In the event the request does not include enough information to derive a site, we arbitrarily made our AM default to our largest site. This keeps Grid'5000 usable for end-users even when used through tools with incomplete support for the component manager concept.

B. RSpec Limitations for Resources Description

Because of GENI's focus on networking, RSpec documents used to describe and request resources are centered around two main elements: nodes and links. Nodes are considered as individual computational elements in the middle of the network being tested. This is a striking departure from the Grid'5000 setup where the cluster a node belongs to is a major characteristic of the node.

1) Resource Description: Advertisement RSpecs describe nodes using a unique identifier, the identifier of the AM they belong to, one or more types of bookings (bare-metal, virtualized, ...), and possibly a list of available disk images. Specification of the actual underlying hardware was added later with the introduction of hardware types. Hardware types are sub-elements added to a node. Depending on the testbed, hardware types can be used to describe the node's make and model (e.g. Emulab's 'nuc8650'), the underlying architecture (e.g. Emulab's 'x86_64'), a generic type (e.g. w-iLab.t's 'LTE-FEMTOCELL'), or a cluster (as in Grid'5000).

The problems with the hardware type mechanism are twofold. First, a simple hardware type cannot convey the full configuration of a node, forcing users to reference external information alongside the *advertisment RSpec*. To remedy this problem Fed4FIRE has designed the *hwinfo*[4] RSpec extension, which allows AMs to add hardware type descriptions to the *advertisement RSpec*. Second, many interesting properties of a node, such as CPU speed or RAM size, are quantitative. This is at odds with the hardware type system of assigning a tag to the node. This use case appears to be filled in part by Emulab's `fd` elements which allows one to specify a value to a name property. However, both `hw-info` and `fd` are not part of the mainline standard, and their availability varies greatly.

Additionally RSpec's focus on a node as an independent unit of computing creates an inefficient organization of the resource description. Disk images, which are not intrinsic properties of nodes, appear as sub-elements of the nodes. This means that a disk image element is repeated on every node it can run on. For Grid'5000, where provided images can run on every available node, this means each disk image is repeated 828 times in a single *advertisement RSpec*.

Comparatively, Grid'5000's *reference API*[5] can provide a node's cluster, CPU (architecture, microarchitecture, speed, caches sizes), RAM size, network adapters (type, addresses, connected switches), bios version, and much more information, in a more compact and human readable JSON format [13]. Moreover most of this information can also be used to filter resources in OAR jobs.

2) Resource Booking: The difficulties with describing resources become a problem when the time comes to request resources. *Request RSpecs* work well in cases where users want to book a specific node or in cases where any node within a testbed would do. However users interested in nodes based on a set of properties or on relations to other nodes will face more difficulties. Hardware types can be used for requests on some testbeds, but Fed4FIRE tools using them only allow one type per node. This makes hardware types suitable to describe the node's complete configuration, but not to be used as a set of piecemeal properties to combine. In Grid'5000, this means hardware types are most suited to indicate the node's cluster, since all nodes in a cluster have the same hardware configuration. Hierarchies and properties as found in Grid'5000, where users can ask for a specific number of nodes distributed over a given number of clusters, fulfilling a complex set of properties using logic operators, cannot be expressed in RSpec. Users are forced to choose the clusters themselves, a decision best handled by the Grid'5000's resource scheduler. These constraints are prejudicial to GENI APIs, because the ability to obtain resources based on properties or relations helps push experimental reproducibility beyond a simple replication on the exact same hardware, which is an important part of experimentation with distributed systems.

Recommendations:

- Explore standardization of resources description in RSpecs.
- Explore standardization of methods for building the *Request RSpec* from the *Advertisement RSpec*.

[4] https://grid5000.gitlabpages.inria.fr/gcf-grid5000-plugin/hwinfo.html

[5] https://public-api.grid5000.fr/stable/sites/nancy/clusters/gros/nodes.json?pretty=1

C. Allocation and Provisioning

1) Two-step request: In the AMv3 API, starting a new node is a two-step process. First, the **allocate** is made with the *request RSpec*. The AM schedules resources and creates corresponding slivers before informing the user. The requested resources are held for a short time and released should the user fail to **provision** the slivers. AMs are told to implement **allocate** as a quick, cheap, and readily undoable operation. Second, users are expected to **provision** the wanted slivers. The resources in the provisioned slivers are supposed to be started at this point. Some testbeds also require the `geni_start` command to be issued using the **performOperationalAction** call. This multi-step process is useful in a testbed federation setup, as the allocation allows a user to test the availability of resources across all the testbeds involved in the experiment with provisioning only happening if a sufficient number of resources is available, and canceling if not. However this setup's efficiency is dependent on whether a testbed can implement **allocate** as a lightweight scheduling. In our current OAR installation, we can schedule resources for future use, but we can not schedule resources for immediate use without starting them. This makes the allocation process in Grid'5000 heavier than expected by the GENI AM API, but in most cases it allows us to start image deployment immediately when users **provision** the slivers.

2) Slice, Slivers, and Jobs: Slices are federation-wide namespaces attributed by the federation *Clearinghouse* to a user, or group of users, for them to run their experiment. With a credential for this slice, the users can ask any AM in the federation to **allocate** resources to the slice. Resources are provided as AM-wide slivers. In Grid'5000, OAR *jobs* are the finest granularity of resources manipulation. Grouping resources into a single *job* is possible, but requires complex RSpec transformations and would bind the corresponding slivers together. Giving each node their own OAR *job* makes the RSpec transformation easier and offers a direct link between sliver status and *job* status. However, although OAR can easily perform multiple single-node jobs, the disk image deployment system does not scale as well and is much more efficient when grouping similar deployments in a single operation. This seems to be a recurrent problem since the AM API offers options to force users to **provision** all slivers in a slice at once.

Conspicuously absent from the AM API is a way to make advance reservations. Some testbeds such as WiLab and BonFIRE accept *geni_start_time* options to an **allocate** call, but this capability is far from standard and there is no information on how such advance reservations should be provisioned. Advance reservations guarantee resource availability for large-scale experiments. The testbeds with advanced reservation capabilities, such as CloudLab [14] and Grid'5000, allow users to perform preparatory work using smaller allocations before doing their large advance reservation for their experiment.

Recommendation:

- Improve support for advance reservations.

D. Resource access

To connect to Grid'5000, users first need to SSH into one of two access gateways. From there they are able to SSH into any site front-end or any node they have booked. The SSH keys used in these connections are managed by Grid'5000's user management system. RSpec provides a method to describe how to connect to a node, which is complemented by the proxy extension designed by Fed4FIRE. Since not every testbed provides users with a home directory shared between the nodes, users might rely on this SSH access to setup and execute their experiment. The **provision** call offers the possibility to inject SSH keys into the provisioned nodes to share access with tools and other experimenters. This works in conjunction with the federation *Clearinghouse*'s ability to issue credentials for a slice to multiple users to create true experiment sharing between researchers. However Grid'5000 is not built in a manner conductive to this kind of experiment sharing and the option to inject SSH keys poses multiple security problems. First, key injection only works in cases where users deploy a disk image, since KaDeploy is the tool setting the new SSH keys to the root account of these images. In cases where users don't deploy an image, they log on using their personal account to the standard Grid'5000 environment and so can only use SSH keys set in their account. Second, even in cases where a key is deployed to a node, connecting into the access gateway requires a Grid'5000 account and can only be done using the keys registered in the user account. Third, in cases where the initial user shares his node with a second user who has a Grid'5000 account, the second user gains full access to the initial user's home directory. Overall, the mechanisms for experiment sharing and SSH key injection do not seem well suited to testbeds that provide per-user storage accessible from reserved nodes, or rely on SSH access gateways for external access.

E. State of the Standard

GENI AMv3 API was finalized in May 2012, and some technological choices, such XML-RPC, have since then lost in popularity in favor of alternatives like REST and JSON. The GENI Engineering Conferences have been considering changes, but no version 4 has yet been proposed. Meanwhile testbeds and federations have worked around the standard by creating RSpec extensions leading to a greater fragmentation of the API.

Recommendations:

- Resume standardization efforts, consolidate extensions.
- Explore more recent technological foundations (REST).

V. Related Work

With SFA being the de facto standard for testbed federation management it pushes testbeds to either adopt GENI stacks or, as Grid'5000 did, adopt a compatibility layer to offer a GENI API. This is clearly visible in testbed federations. One could argue that this leads to some ossification in the context of testbed control interfaces, as shown by the limited number of alternatives.

Some testbeds still expose different control interfaces – some of which were created during the early exploratory phase of the GENI project. The ExoGENI subproject built a distributed Infrastructure-as-a-Service (IaaS) testbed where resource control was operated by an out of the box cloud management system on top of which a testbed management system based on Open Resource Control Architecture (ORCA) [15] was deployed. In ORCA APIs, resources are described with extensive sets of properties, and brokers could be used as intermediaries between users and ORCA AMs. ExoGeni testbeds also provide GENI AMv2 API. The ORBIT testbed concurrently developed the cOntrol and Management Framework (OMF) [16] offering not only an interface for resource management but the management and monitoring of whole experiments. In OMF a resource request can be performed using a programmed description. This concept is also used in Cloudlab [9] although the programmed description generates a standard *request RSpec*.

Other testbed management stacks are more recent. Tsumiki [17] is a testbed management system framework in which operators are invited to build testbeds suiting their needs. WalT [18] is designed to work with cheap hardware therefore allowing any user the ability to easily reproduce both the testbed and the experiment running on it.

The ChameleonCloud [8] testbed leverages the IaaS control software, Openstack, to manage its resources, and adds additional tools and services. This allows users to book resources using the standard Openstack clients. Similarly EdgeNet[6] is a nascent testbed for geographically distributed experiments, in the vein of PlanetLab, leveraging the Kubernetes container orchestration software.

VI. Conclusions

In this article we presented the implications of implementing the GENI AMv3 API in the Grid'5000 testbed. We draw several lessons and recommendations from this work and a comparison of Grid'5000 and GENI capabilities. The main area of improvement we perceive in GENI APIs is the description and selection of resources, which lacks support for advanced filtering or more abstract nodes relationships. Research on this matter is already ongoing and other teams participating in the Fed4FIRE+ project are looking to build a RSpec extension based on ontologies.

A popular direction for recent testbeds is to build their services on industry-grade software like OpenStack and Kubernetes. It will be interesting to understand how this affects, or not, the ability to integrate with a standard API, or how lessons from those widely adopted infrastructure management frameworks could influence the testbed community. Having an interface that can represent the resources and topologies needed for an experiment in a high-level and precise fashion and that can provision and orchestrate these resources anywhere in the world would allow for unprecedented levels for experiment reproduction at a time when computer science is facing a reproducibility crisis.

[6]https://edge-net.org/

References

[1] B. White, J. Lepreau, L. Stoller, R. Ricci, *et al.*, "An integrated experimental environment for distributed systems and networks," in *OSDI*, 2002.

[2] B. N. Chun, D. E. Culler, T. Roscoe, A. C. Bavier, *et al.*, "Planetlab: An overlay testbed for broad-coverage services," *Computer Communication Review*, 2003.

[3] D. Raychaudhuri, M. Ott, and I. Seskar, "ORBIT radio grid tested for evaluation of next-generation wireless network protocols," in *TRIDENTCOM*, 2005.

[4] T. Benzel, R. Braden, D. Kim, C. Neuman, *et al.*, "Experience with deter: A testbed for security research," in *TRIDENTCOM*, 2006.

[5] M. Berman, J. S. Chase, L. Landweber, A. Nakao, *et al.*, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, 2014.

[6] R. McGeer, M. Berman, C. Elliott, and R. Ricci, *The GENI book*. Springer, 2016.

[7] D. Balouek *et al.*, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, 2013, pp. 3–20.

[8] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, *et al.*, "Chameleon: A scalable production testbed for computer science research," in *Contemporary High Performance Computing*, CRC Press, 2019.

[9] R. Ricci, E. Eide, and C. Team, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *;login:*, vol. 39, no. 6, 2014.

[10] L. Nussbaum, "Testbeds Support for Reproducible Research," in *ACM SIGCOMM 2017 Reproducibility Workshop*, Los Angeles, United States, 2017.

[11] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, *et al.*, "A batch scheduler with high level components," in *IEEE CCGrid 2005*, 2005.

[12] E. Jeanvoine, L. Sarzyniec, and L. Nussbaum, "Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters," *USENIX ;login:*, vol. 38, 2013.

[13] D. Margery, E. Morel, L. Nussbaum, *et al.*, "Resources Description, Selection, Reservation and Verification on a Large-scale Testbed," in *TRIDENTCOM*, 2014.

[14] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, *et al.*, "The design and operation of cloudlab," in *USENIX Annual Technical Conference*, 2019.

[15] J. S. Chase and I. Baldin, "A retrospective on ORCA: open resource control architecture," in *The GENI Book*, R. McGeer, M. Berman, *et al.*, Eds., 2016.

[16] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "OMF: a control and management framework for networking testbeds," *Operating Systems Review*, 2009.

[17] J. Cappos, Y. Zhuang, A. Rafetseder, and I. Beschastnikh, "Tsumiki: A meta-platform for building your own testbed," *IEEE Trans. Parallel Distrib. Syst.*, 2018.

[18] P. Brunisholz, E. Dublé, F. Rousseau, and A. Duda, "Walt: A reproducible testbed for reproducible network experiments," in *CNERT*, 2016.