Interoperability in Federated Clouds

by

Sushil Bhojwani

B.E., University of Rajasthan 2008

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Interoperability in Federated Clouds

by

Sushil Bhojwani

B.E., University of Rajasthan 2008

Supervisory Committee

Dr. Yvonne Coady, Supervisor

(Department of Computer Science)

Dr. Sudhakar Ganti, Supervisor

(Department of Computer Science)

Supervisory Committee

Dr. Yvonne Coady, Supervisor

(Department of Computer Science)

Dr. Sudhakar Ganti, Supervisor

(Department of Computer Science)

ABSTRACT

Cloud Computing is the new trend in sharing resources, sharing and managing data and performing computations on a shared resource via the Internet. However, with the constant increase in demand, these resources are insufficient. So users often use another network in conjunction with the current one. All these networks accomplish the goal of providing the user with a virtual or physical machine. However, to achieve the result, virtual machine users have to maintain multitude credentials and follow a different process for each network. In this thesis, we focus on SAGEFed, a product that enables a user to use the same credentials and commands to reserve the resources on two different federated clouds, i.e., SAVI and GENI. As a part of SAGEFed, the user can acquire or reserve resources on the clouds with the same API. The same service also manages the credentials, so they do not have to manage different credentials while acquiring resources. Furthermore, SAGEFed demonstrates that any cloud that has some form of client tool can be easily integrated.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank my supervisors **Dr. Yvonne Coady and Dr. Sudhakar Ganti** for their support and mentoring throughout the degree. I would also like to thank **Dr. Rick Mcgeer** for presenting this opportunity and providing technical advice and assistance.

Finally, I would like to thank my friends and family for their support and patience.

DEDICATION

To my Parents and my mentor and friend, my Brother Sunil.

Chapter 1

Introduction

The key to growth is the introduction of higher dimensions of consciousness into our awareness. –***Lao Tzu***

Distributed applications are enhancing and omnipresent in our everyday lives. Viewing a film on-line, forwarding an email and conversing with friends over the Internet are uncomplicated cases of Distributed Computing. All these applications use the distributed network across the world.

Throughout the distributed system, users are often required to run experiments on different research testbeds, or clouds. Every time users need to access a new cloud, they need to learn and then use a cloud-specific set of instructions to reserve resources, for example, a virtual or physical machine, and release the resource at the end of the experiments. It is essential to release the resource, as those testbeds have defined resources that are shared among a large number of people. To access various testbeds, we need separate credentials. This scenario is not rare. Using different email ID's from several hosting servers is a common example. The goal is the same in every case, for example, sending or receiving an email. However, to do this straightforward task users have to become familiar with the different GUI (Graphical User Interface)

options or commands and manage multiple credentials.

Testbeds like SAVI (Smart Applications on Virtual Infrastructures) [30] and GENI (Global Environment for Network Innovations) [12] are essential to network researchers' work. However, testbeds are limited in resources. The resources are required not just by the networking researchers but also by researchers from other scientific areas to perform cumbersome computations. Another factor is that it is not possible for an individual network company to establish the network in the main cities in every country. To make better use of these research testbeds, they need to share resources. Sharing resources will not only scale the cloud but can also be useful for many distributed applications. The Pollution Visualizer [39] presents an idea of collaborative work over distributed network. The pollution information would be hosted on multiple virtual machines all over the world. Distribution of data would help to reduce the latency while processing the data which, in turn, will enhance the performance. Celesti [14] proposed a three phase model to enable cross-federation. Most of the current research in this area is focused on improving the architecture, but did little research to make the federation user-friendly using the existing cloud architecture.

Now a days almost all the applications use a single sign-on (SSO) [20] service that allows users to use their Facebook or Google account credentials to login to their services. Also, many companies try and keep the GUI format alike. Both of these approaches help users. In the first case, users can bypass a situation where they must remember one more password and in the latter case, the user are protected from the effort of understanding everything again.

SAGEFed is a lightweight Unix based command line script, and acts as a wrapper on top of client tools such as Nova [43] and Omni [45]. Rather than providing a new client tool that operates only with GENI and SAVI, it is designed to wrap around any existing client tools, thus introducing minimal overhead. The advantage of SAGEFed

is that it can support any number of client tools under a common API. Users do not need to use the specific client tool commands. Instead, they can use the SAGEFed wrapper to obtain and release resources on the multiple testbeds. Various client tools have different commands and terms. SAGEFed allows users to use the client tool commands directly, in addition to those provided by SAGEFed. If users do not want to learn complex commands, then they can use same API for various client tools. In this way, it offers flexibility to use client tools and also provides a common flexible and extensible API.

1.1 Cloud Computing

The computer utility can become the basis of a new and important industry. John McCarthy at MIT Centennial Talk in 1961. [15]

Cloud computing provides scalable computing and storage resources via the Internet. Cloud computing as a whole was introduced by Eric Schmidt in 2006. A cloud is a combination of physically distributed computer resources. These resources are ordered on-demand to fulfill the needs of the user. In this way, the cloud can grow or shrink in real time [57]. The main purpose of the cloud is to provide scalable computing and storage resources through the Internet [4, 16, 18]. These resources are usually hosted on a network that is accessed using the Internet. Users have become more sensitive to the network issue related to services they use, and slow Internet speed is one of the common examples. Hence, to maintain these real-time demands, the cloud needs to allocate more resource and these extra resources, can be freed back into a pool once the peak period is over. Figure 1.1 provides an overview the basic architecture of cloud computing.

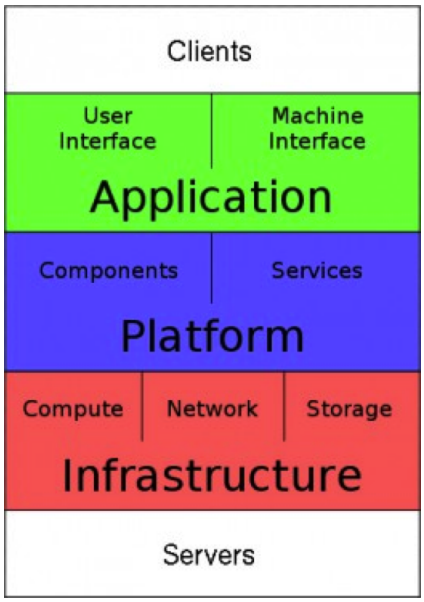


Figure 1.1: Basic Cloud Computing Architecture[47]

1.2 Openstack Components

OpenStack [59] is an open-source cloud software that was launched by NASA and Rackspace in July 2010. The main aim of this project was to offer cloud-computing services over standard hardware. Omar [59] discussed key features of OpenStack, which include properties such as scalable, compatible and flexible. OpenStack has gained a reputation as a reliable, robust IaaS (Infrastructure as a Service) solution for public and private cloud infrastructure [17]. Figure 1.2 represents the conceptual architecture of OpenStack with all the major components and the interface between them. By default, OpenStack uses MySQL [41] to store management and configurational information.

Nova [43] is a cloud computing fabric controller and is the heart of OpenStack. It was built to control the available resources and share the global state of the cloud. Nova drivers communicate with the different hypervisors that run on host operating systems and reveal functionality over the Internet. Nova provisions the requested

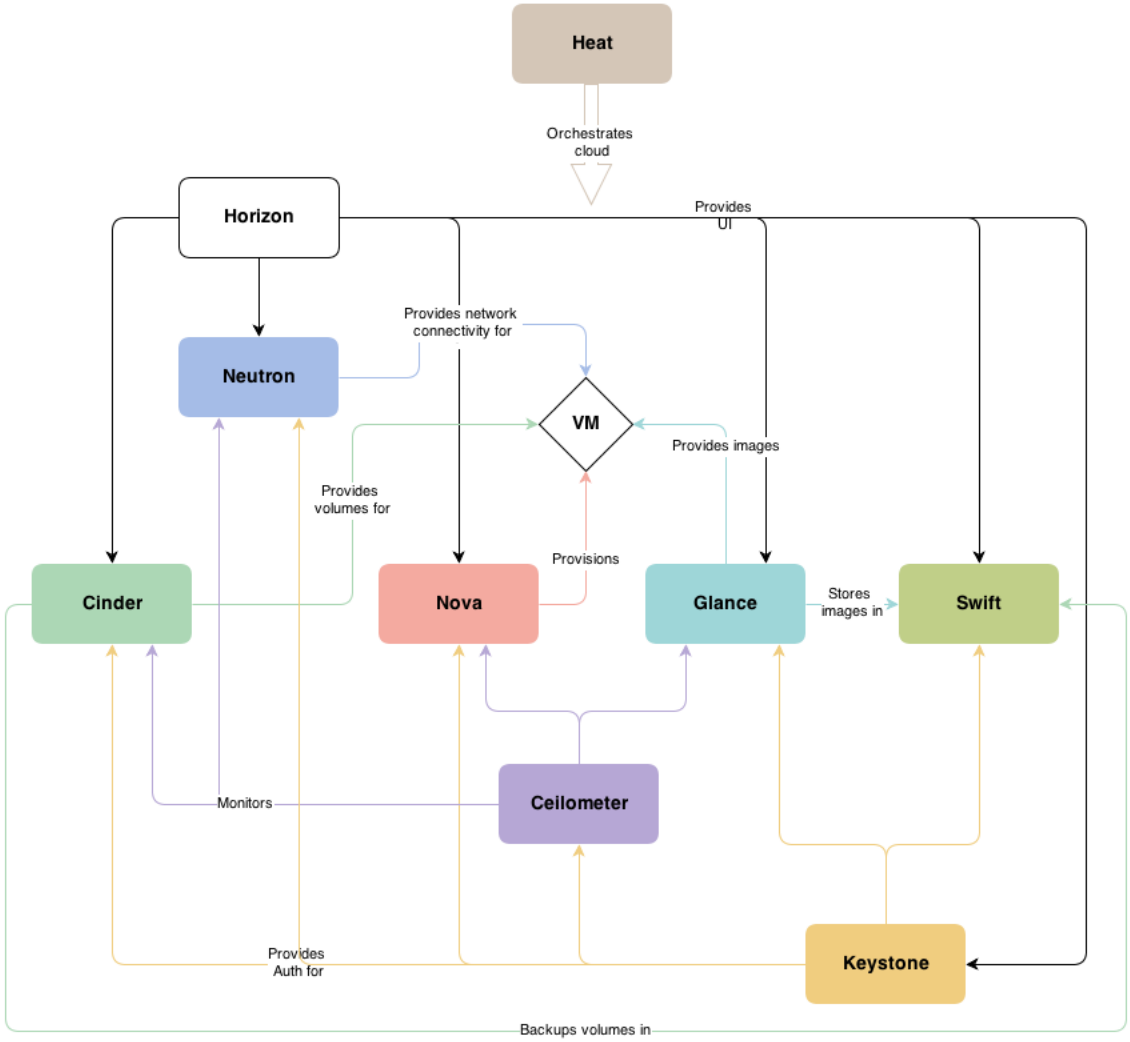


Figure 1.2: Openstack Conceptual Architecture [28]

virtual machine with the help of information provided by other components such as cinder and swift. The command line client tool is also available for users who want to create a virtual machine or want to integrate resource management with their software. The client tool uses the configuration file that stores user credentials and addresses of the Nova server. Once the user executes any command, the call is sent to Nova server, and respective action is taken.

Glance is a collection of virtual machines (VM) images. Nova coordinates with

the Glance through an API. The API queries the image metadata and retrieves the demanded image for the creation of VM. It is one of the essential services of basic cloud computer architecture.

Neutron is a networking service that not only provides the network connectivity among the various services of OpenStack but also manages the topologies, advanced policies along an interface between the various VM and allotting them IPs.

Swift is widely available and is known as OpenStack object storage. The primary role of Swift is to allow users to store and retrieve data. For other services, Glance uses Swift to store all the images and Cinder, on the other hand, backs up all the VM's volumes.

Cinder provides the volume to VM and store those volumes as a backup in Swift. It specifies the number of functionalities such as creating, deleting and attaching volumes. It also supports the creation of snapshots of the volumes which can then be used to establish a clone of the VM.

Keystone works as an immigration officer and validates the identity of the users. It supports token-based authentication. All other components are dependent on the keystone to validate any credentials. Recently it has been re-architectured to support external services such as oAuth, SAML, and openID.

Horizon is commonly known as Dashboard for OpenStack. Horizon interacts with all the components and presents the information in a systematic manner on a web application for users.

1.3 Introduction to GENI

GENI (Global Environment for Network Innovations) is an innovative suite of infrastructure to support experimental research [19, 36]. The goal of GENI is to enhance

experimental research in distributed networks and build the services. These services will not only improve the economic growth of the United States but also provide an environment for researchers to perform high scale computations. Researchers can acquire computer nodes, mobile sensors and wireless sensors through GENI.

A Project is a collection of people and experiments. The project lead handles the creation and all the activities related to that project. A project leader can be connected with multiple projects while multiple experimenters can be linked with one project.

A Slice allows multiple experimenters are related with one project and may be running multiple experiments at a time so to avoid any conflicts, slice was introduced. A slice is an independent isolation for experimenters where they can run their experiments. While creating the slice, the experiment can provide access to other experimenters in the project. The project leader is a member of all the slices in a project by default.

A Sliver is an independently managed resource in a slice. They can also be termed as Virtual Machines. Slivers are the place where researchers can perform their experiments.

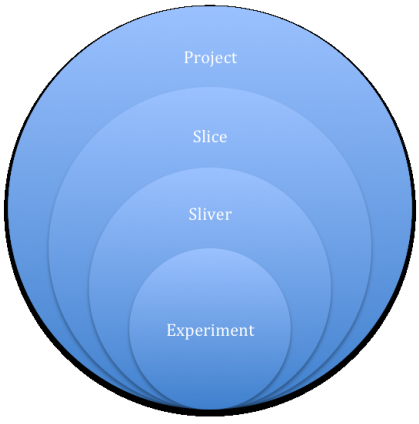


Figure 1.3: GENI Experiment Structure

GENI Aggregates are the physical hardware that provide the desired resources to the experimenters. For example, a Rack at the University of Utah is an aggregate.

RSpecs are XML's (EXtensible Markup Language) in the predefined format used to identify the type of resources requested. The experimenter specifies the RSpec while acquiring resources and sends this to Aggregate Manager (AM) in a request. The AM validates the RSpecs and then allocates the resource.

1.4 Introduction to SAVI

SAVI (Smart Applications on Virtual Infrastructures) is a research testbed with the purpose of creating and delivering future Internet applications [31, 30]. It provides infrastructure to support experimental research in cloud-computing, advance media processing, etc. The goal of the SAVI network is to help grow Canada's Digital Economy Strategy by providing infrastructural support to innovative applications. It provides access to computing blades, Networking and GPU's. The SAVI testbed is an OpenStack system. Figure 1.4 shows the basic architecture of SAVI.

Tenant is a group where experimenters belong to the same organisation are grouped together. Grouping users on the basis of organisation is a better way to restrict the user involvement from other organisations. For example, users from the University of Victoria are grouped under "uvic" and users from the University of Toronto are grouped under "UoT".

Edge is the rack location where all the physical hardware is present. The experimenter needs to define the Edge name while requesting resources. All the Edge names are unique. Once the demand is submitted, SAVI identifies the rack and allocates the resources to the experimenter. All the experimenters from same tenant group can view all the resources allocated on any edge.

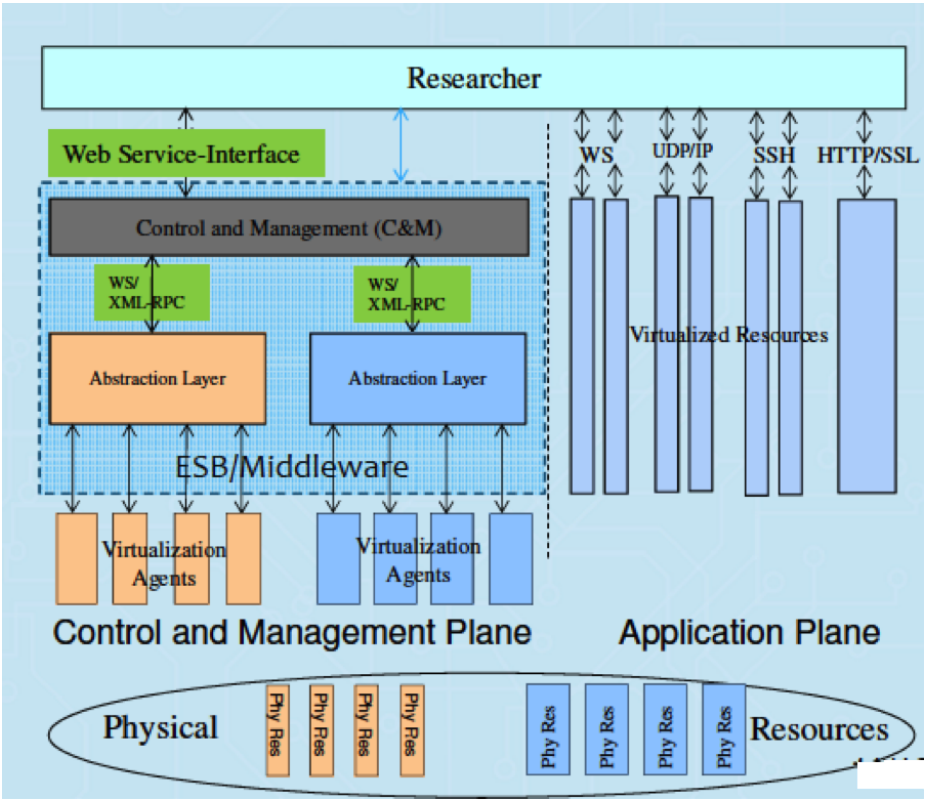


Figure 1.4: SAVI Architecture [28]

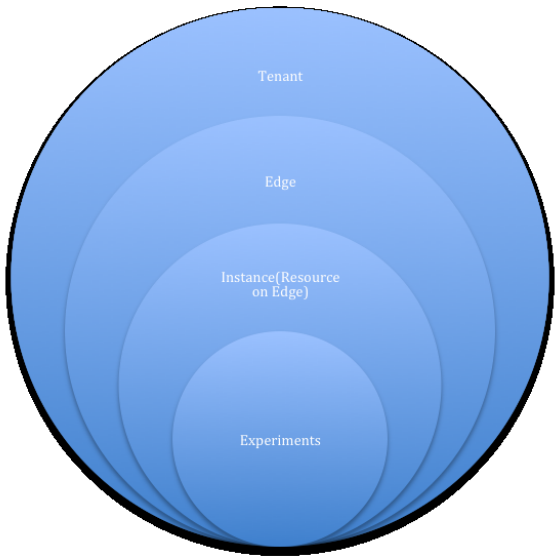


Figure 1.5: SAVI Experiment Structure

Instance are the resources assigned to the experimenter. These can be VMs or physical machines where the experiments are executed. The creator can then set the permission and restrict the other user to enter another instance.

There are many other network testbeds available for researchers. Not all of them have the same architecture, but they all offer the identical service of providing resources for experiments. PlanetLab [9, 50], CoreLab [42], Emulab [65], OneLab [46] and VINI [8] are few other testbeds available for research experiments.

1.5 Motivation

The goal of SAGEFed is to allow a federation of resources provided by GENI with SAVI or any other testbed and reserving them through the common API working with same user credentials. When SAVI is federated with GENI, all its resources are readily available to GENI users and vice-versa. This not only enhances the scalability of the cloud but also prolongs its presence in a different part of the world such as Canada and the USA. Application developers are moving from conventional platforms to clouds to benefit from a distributed infrastructure. Take a project where specialists from various parts of the world (USA and Canada) want to work together and analyze health care data. These datasets are too big to be stored on a local desktop. Such applications are sensitive to network latencies and storing data at one location is going to produce high latency, when various users try to access the data. Instead the data collected can be stored in different instances in the cloud. When the services need to analyze the collected data they can access the data from the nearest instance.

Federation of clouds provides many benefits, but it has a few drawbacks. It is currently lacking in terms of usability and the interoperability of federated clouds. Acquiring resources can turn into a cumbersome task of learning new commands and

maintaining new credentials for different clouds. SAGEFed fills the gap by providing common API for resource management and also providing an authentication mechanism where the same credentials can be used to enter the other cloud.

The remainder of the dissertation is organized as follows, Chapter 2 gives background and related work on different federation approaches. We also look at benefits of federation and interoperability in cloud computing . Chapter 3 gives the architecture of SAGEFed. I outline the design goals of SAGEFed and discussed about the different components in detail. Chapter 4 details the prototype implementation of SAGEFed. I discuss how it interacts with the system, as well as how applications can use SAGEFed. Chapter 5 gives experimental results with SAGEFEd. The results report on the performance and user experience with benchmarks. Chapter 6 has concluding remarks as well as some directions for future work.

Chapter 2

Related Work

You have to learn the rules of the game. And then you have to play better than anyone else. –***Albert Einstein***

In the early lifecycle of cloud-computing, federation was not considered to be a first class problem. However, with the progression of the network and computers, there was an increase in demand for more resources. The cluster administrator has a few options to fulfill the request of resources: buy more hardware, direct users to the bigger service provider or combine a few mid-sized batches. The third option of combining few mid-sized batches is cheaper and provides more resources. The process of combining the clusters is known as *federation*. This chapter provides an overview of the various federation architectures and approaches for providing authentication. We also discuss the benefits of applications over the cloud.

2.1 PlanetLab and Wireless NITOS federation

Wireless Mesh Networks (WMN's) [49] are a default solution to providing high-speed Internet. The federation between two heterogeneous network testbeds, i.e., NITOS (Network Implementation Testbed using Open Source platforms) wireless and the

PlanetLab Europe (PLE) wired network was achieved with OMF (ORBIT Management Framework) [53] a common experiment control framework and slice abstraction as a building block. Stratos et al. [34] presented a framework which provides single sign-up so that common credentials can be used. To overcome the difficulty of the different languages to describe the resources configuration and action, the PLE architecture was changed to incorporate OMF on demand. If any slice created with this tag was activated, the OMF resource controller was installed on that particular slice. Hence, PLE resources can be viewed as any other resources of an OMF-based testbed.

2.2 Three-Phase Cross Cloud Federation

Celesti [14] presented a three-phase design to achieve cross cloud federation. The main component of conception was the discovery phase, match-making phase and the authentication phase. The discovery phase is to find a foreign cloud and broadcast the resource state and supported features to other peers through a peer-to-peer [63] approach. The discovery agents can get the information about other clouds by querying the shared location. In the match-making phase, the information collected from other discovered clouds is classified into quantifiable and unquantifiable. Once the classification is done, the best foreign cloud is selected on the basis of the numerical analysis. The final phase is the authentication phase where a federation is established between the home cloud and the foreign cloud. The SSO [66] authentication process needs to be started by the home cloud. Shibboleth [40], OpenID [55], and SAML [29] can be used to address the SSO authentication problem.

2.3 Other Federation Approaches

Kurze [35] provided a reference architecture for cloud federation. The architecture has different components where a provisioning engine matches the business logic components to a pool of resources. The distribution manager, with other sub-components, enforces consistency between data replica and the same configuration over multiple servers. In the end, the resource manager manages all the resources in a centralized way and can be considered a resource pool. Figure 2.1 represents the reference architecture. Grid computing [11] can be seen as an extension to cloud computing where the resources are geographically distributed and managed by individual administrative domains. However, Rajiv [54] discussed various problems such as security, resource discovery and policy involved in the federation. GridShib [64] came up as a solution to deal with the federation problem. GridShip is a combination of Shibboleth and the Globus Toolkit [23]. When users want to use the Globus to access the grid they send their X.509 certificate, and Grid contacts the Shibboleth service for authentication. Tom [6] and Xinming [48] have proven that the Grid works with several existing grids.

Omni [21] is a leading prototype for flexible federation. In this federation, all the testbeds are based on PlanetLab's [52] code base. All the other clouds such as PlanetLab and EmanicsLab, run PlanetLab software stack and work under the same API by default. On the other hand, GENICloud [7] works on a Slice-based facility architecture (SFA) [51]. Only one extra software layer is required to make the other clouds SFA compatible. Eucalyptus [44] is one of the basic examples where no concept of the slice is present. Another popular approach is the service-oriented architecture [10] that allows integration between business services and the multiple domains across the world. Most of the federation technologies orbit around federating identities from

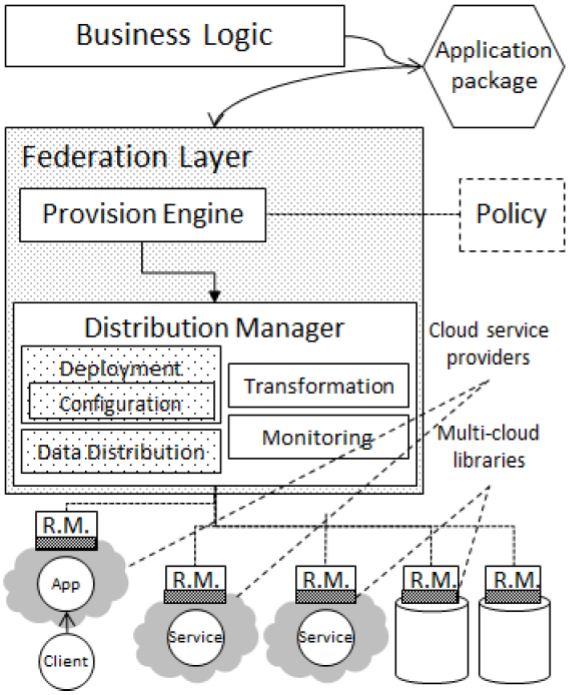


Figure 2.1: Reference Architecture [35]

different security domains. Jun and Alan [37, 32] pointed out the demerits of identity approach and proposed a new authorization-based access control [32, 33]. In the authorization-based approach, the security domains manage just their users. Service names Zebra Copy [38] is the prime example for authorization-based access control and is available for use. Sadashiv and Kumar [56] compared the cluster computing [13], Grid Computing and Cloud Computing [5] and the key challenges they faced. Conceptually all seem to have similar features. However, they choose grid and cloud computing as a promising model for interoperability and focus on standardizing API. In another comparison between cloud computing and grid computing, Foster [24] proposed a new architecture and agreed that grid and cloud computing are very similar but cloud still has a greater advantage in running an application. Dikaiakos [18] has discussed vision to see all clouds as one and key challenges to achieve the same. Cloud interoperability can change the way we see the clouds. They will be

amended in a collection of the transparent platforms where applications will not be restricted to the specific cloud. It would be like an open place where any application can use any resource on any cloud.

2.4 Benefits of Federation and Interoperability in Cloud Computing

With the exponentially escalating data and computational power, the world is looking for cheaper hardware and computing options. Schadt et al. [58] discussed how the open cloud and heterogeneous computing could be used to tackle big data problems. The CloVR application [3] uses cloud-computing platforms that provide on-demand access to the computing infrastructure. The use of cloud computing helps them to achieve high throughput data processing at low cost. Fusaro and Patil [25] exhibited an example of how the Amazon services such as AWS (Amazon Web Services) can be used for computational problems and how the other applications can take advantage of the flexible cloud resources. Even the education sector is not behind in taking advantage of cloud computing. Faisal [2] compared various educational cloud-computing applications and concluded that the education sector is taking advantage of the services on the main parameters of security, reliability and portability. A detailed case study [60] was presented to show the results of these experiments. Gupta [27] evaluated how the HPC applications can be best utilized on the cloud. Their results showed that the cloud a is viable platform for the application where we have tree-structured and communication-intensive applications.

2.5 Summary

This chapter discussed various approaches towards federation and the need for cloud computing in different fields. With the growing needs of users, the demand for a flexible distributed system and a more efficient federated distributed system has developed. Most of the federation techniques involved the change in the underlying architecture. SAGEFed has a slightly different approach to federation that makes federation and interoperability easier to use, while still being highly extensible to future clouds. In the next chapter we will take a deep dive in the architecture of the SAGEFed and its different components.

Federation	Federation Approach	Changes
PlanetLab & NITOS	Orbit Management Framework	Architecture changes to adopt OMF.
Three Phase Cross Cloud federation	Three Phase approach	New architecture with discovery, match-making and authentication
Other Federation	New federation layer	Change in underlying architecture to adopt new system

Table 2.1: Summary of different federation approach

Chapter 3

SAGEFed Architecture

Architecture is the reaching out for the truth – ***Louis Kahn***

In this chapter, we will present a high-level view of SAGEFed. Once we understand the high-level overview, then we will take a deep dive into each component for more details. Finally, we will see few of the possible extensions in SAGEFed and how we can incorporate them into the architecture in future.

3.1 Desgin Goals

The SAGEFed was designed to acquire and release resources on GENI and SAVI. While designing the architecture, the entire focus was on the three elements: simplicity, flexibility, scalability. Simplicity is achieved by making sure we add minimal overhead on the top of the existing system and that anyone can use the tool easily. In order to ensure this, SAGEFed commands resemble descriptive plain-language commands. Secondly, as SAGEFed acts as a wrapper over the clients tools in both GENI and SAVI, the goal was to give user the flexibility to use either the SAGEFed or corresponding client tool commands. Finally, it was aimed to ensure that the tool can be quickly extended to accommodate any other client tool.

Below are the design goals to achieve the elements mentioned earlier:

- Propose a minimal overhead as desirable as corresponded to directly using the client tool.
- Support user flexibility to utilize both client tool commands and SAGEFed commands.
- Let other client tools be integrated quickly in future.

3.2 Overview

SAGEFed is a shell script application. Shell script interacts with the client tools to secure resources. SAGEFed translates the commands to understandable client format, which in turn requires no modification of client tools. Credential management is an essential part of the system. Figure 3.1 represents the fundamental architecture of SAGEFed and its major components:

- Credential management
- Validator
- Translator
- Backend client tools

From the users point of view, SAGEFed is one single component. Internally SAGEFed can be connected to various client tools. The users interact with SAGEFed to list, acquire, and release resources on testbeds. SAGEFed works like a normal command line tool that depends on the arguments supplied to it, and functions accordingly. The supplied command is converted into the client tool understandable format, which is hidden from users.

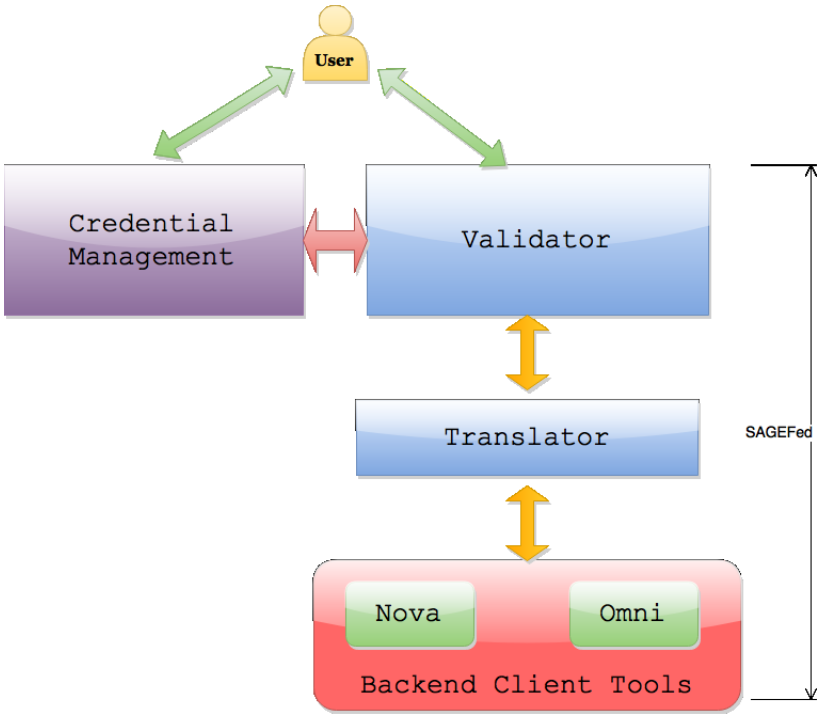


Figure 3.1: SAGEFed Architecture

3.3 Credential Management

In this section, we will look into the authentication and authorization mechanism in SAGEFed. For SAVI, the users need to register before they get a user-name and password. The administrator then approves the request and sends the credentials to the users in an email along with the tenant name (as discussed in section 1.4). Once the users have received the credentials, they can use the same credentials to set up the Nova configuration file.

Similarly for GENI, the users request the credentials. However, the process in GENI system is a bit more complex. Once the users receive their credentials and have linked with some project, they will access the web-interface. On the web server, the user needs to upload a private and public key pair, or he can generate these at the same instant. After all these steps have been followed, the user will download an

Omni bundle. The bundle is a combination of the private key and a certificate that is required to use the command line tool. Omni is the command line tool that is used to do processing on GENI instance.

SAVI-GENI Authentication Architecture

GENI and SAVI systems are both modified versions of the OpenStack Cloud system. SAVI and GENI users have different processes for getting authenticated. As discussed previously in section 1.4 we saw how SAVI and GENI users can get access to the respective tenants and projects. For federation, the SAVI users should have access to GENI instances and vice-versa. Figure 3.2 shows the SAVI-GENI authentication architecture.

How SAVI Users Can Access GENI Resources

For SAVI users to access GENI resources using SAGEFed, they must first download the Omni bundle. The SAVI users visit the GENI portal from where they get redirected to the SAVI Shibboleth Identity Provider (SSIDP) server. SSIDP verifies the SAVI users with credentials stored in the LDAP database. The LDAP database is in sync with the Keystone database that saves all the login information of SAVI users. Once the SSIDP verifies the credentials, the users are redirected to the GENI portal with appropriate certificates. Now the users can download the Omni bundle that stores their credentials. SAGEFed, in turn, uses the Omni tool to access the GENI instance.

How GENI Users Can Access SAVI Resources

If GENI users need to access SAVI resources, they must login to the GENI portal that redirects them to the SAVI portal. The request to the SAVI server includes a certificate that is verified by the SAVI server. After the check is completed, SAVI then

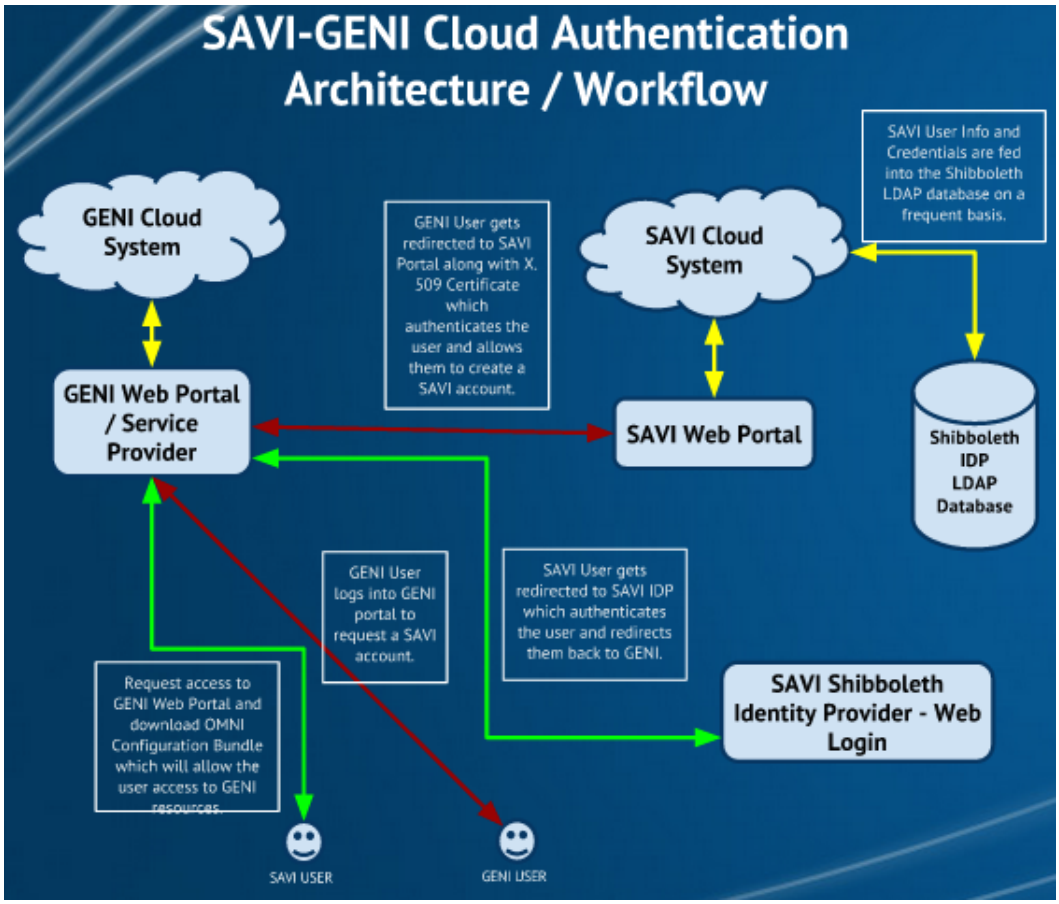


Figure 3.2: SAVI-GENI Authentication Architecture

allows the GENI user to create a SAVI account. This stores the GENI credentials in the SAVI keystone database to enable access to the account in future. In the future when GENI users try to access the SAVI resource, the user is validated against the credentials stored in Keystone database. These credentials are also sent to the users through an email that can be used later to configure the Nova configuration file. After the authentication process, users are forwarded to the SAVI portal through which they can access SAVI resources. SAGEFed uses the Nova client tool to access the SAVI instance.

3.4 Validator

Validator acts as an immigration officer at a border. The main objective of a validator is to check if all the arguments supplied by the user in the command are valid, similar to an immigration officer who checks all the documents before permitting the entrance to a person. The validator has a list of functions that can be performed by SAGEFed. As soon as the command is executed, the validator first checks the function that is the first argument supplied by the user. If the function resembles any of those present in the list, it moves ahead and checks other parameters. The next task is to validate the instance name. As of now, SAGEFed supports only two cases, i.e., GENI and SAVI. In future, with a small modification we can evolve more cases and add them to the list of the validator. The validator is designed to handle the case sensitivity and show an appropriate error message when some invalid function is entered. Figure 3.3 demonstrates the processing of a user command.

3.5 Translator

The translator is the heart of the whole system. Below are the chief duties performed by the translator.

- Identifying the instance
- Modifying the Command to readable client format
- Executing the altered command
- Assembling execution logs
- Providing only relevant information from the logs

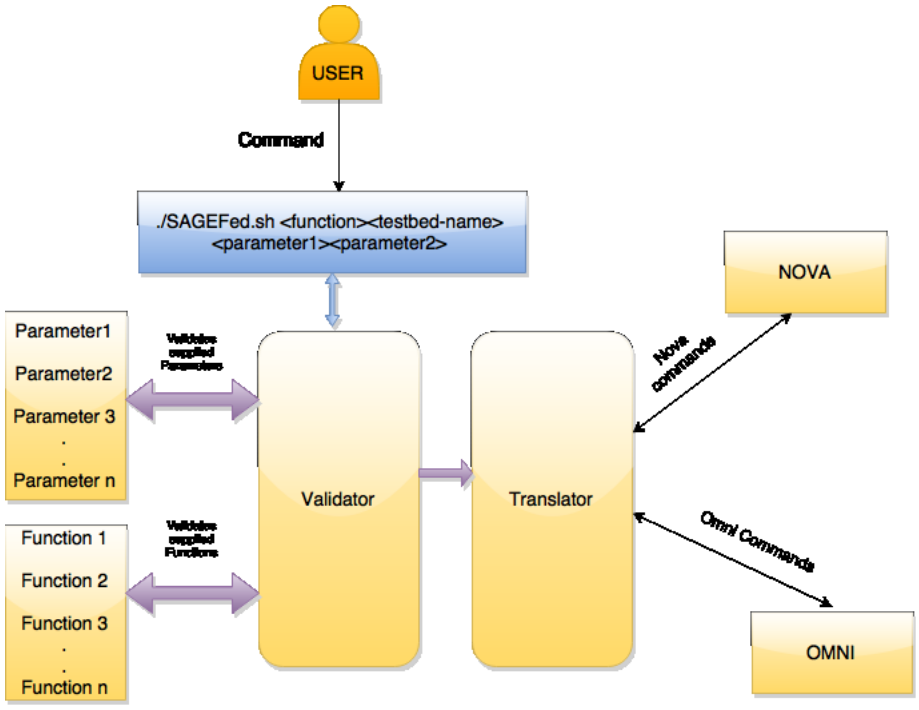


Figure 3.3: Internal Working of SAGEFed

Once the validator clears the command, it is sent to the translator for further processing. The primary objective of this module is to identify the instances where the user wants to perform the actions. Such identification is done by the arguments provided by the user. Once the instance is detected, the translator converts the user command to understandable client format and then executes the translated command. It then obtains the valuable information from the execution logs and presents it to the users. All the logs are stored in the logs and temporary folders, that can be used to fetch more data in future.

3.6 Backend Client tools

This component is a collection of the tools that are available for users to access SAVI GENI instances. As of now we have just two client tools but the backend client

tools component Figure 3.1 can accommodate other tools with minimum changes to the current system. Once the translator converts the user commands into an understandable client format, they are passed to this backend client tools block. The client tool then accesses the configuration files and completes the user's request on the particular instance. Before going further, we need to understand the primary working of these client tools.

3.6.1 Nova-Client Tool

The Nova client tool is a python binding and is a combination of Python API (also known as Nova client) and a command line script (Nova). The Nova client tool is a command line tool that can be installed on any machine and provides access to the OpenStack cloud. Once this package is installed, we get shell command Nova. The various commands are available to perform the several functions on the OpenStack cloud instance. Figure 3.4 shows the internal communication of Nova.

The Nova API interacts with every separate component and performs the desired actions. All the login details such as user-name, password, authentication URL, tenant name and region name are stored in the configuration file. When any Nova command is executed, the Nova API first contacts Keystone to validate the credentials. After a successful verification, Keystone sends an authentication code that can be used by Nova for authenticity to interact with other components until the new command is executed.

3.6.2 Omni Tool

Omni is a GENI command line tool that is used to reserve the resources at GENI aggregate managers (AM) using GENI AM API. It also communicates with clearing-houses and the slice authority to create and delete slices. It is again a Python code

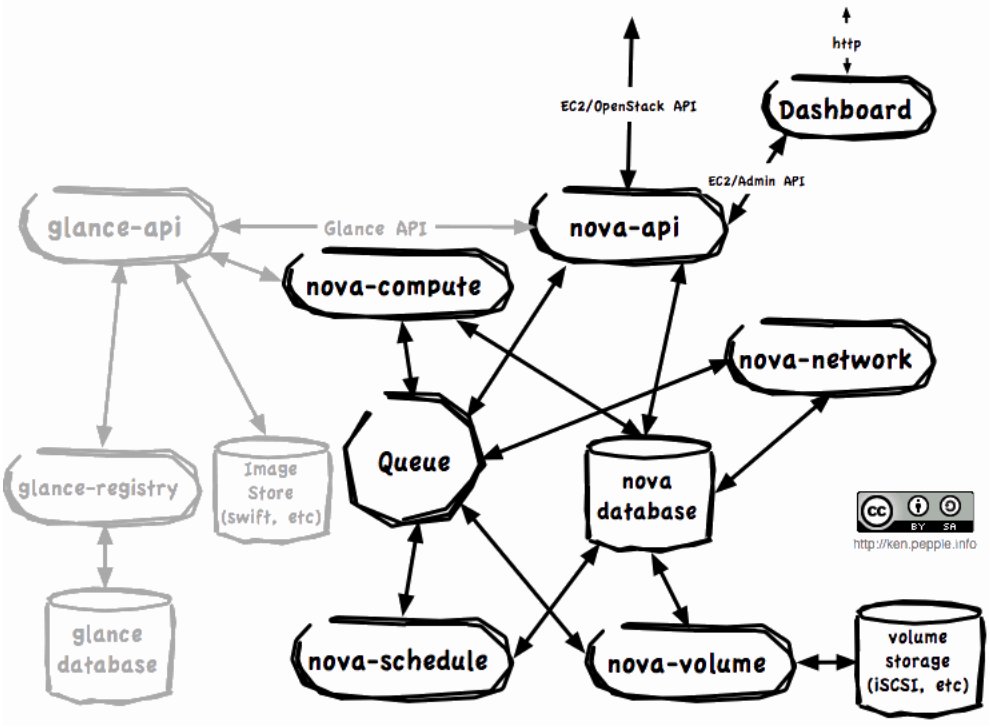


Figure 3.4: Nova Internal Architecture [28]

base that sits over the OpenStack structure. In Figure 3.5 we can see the communication architecture of Omni with AMs and in turn with the underlying OpenStack system. Any GENI user can use the Omni client tool to access the instance through command line.

The users first need to download the Omni bundle that fundamentally stores all the credentials and certificates for authentication. Once the user executes any command, the details with Omni are sent to GRAM (GENI Rack Aggregate Manager) for verification. After the audit, the GRAM interacts with different OpenStack structures to process the command executed by the user.

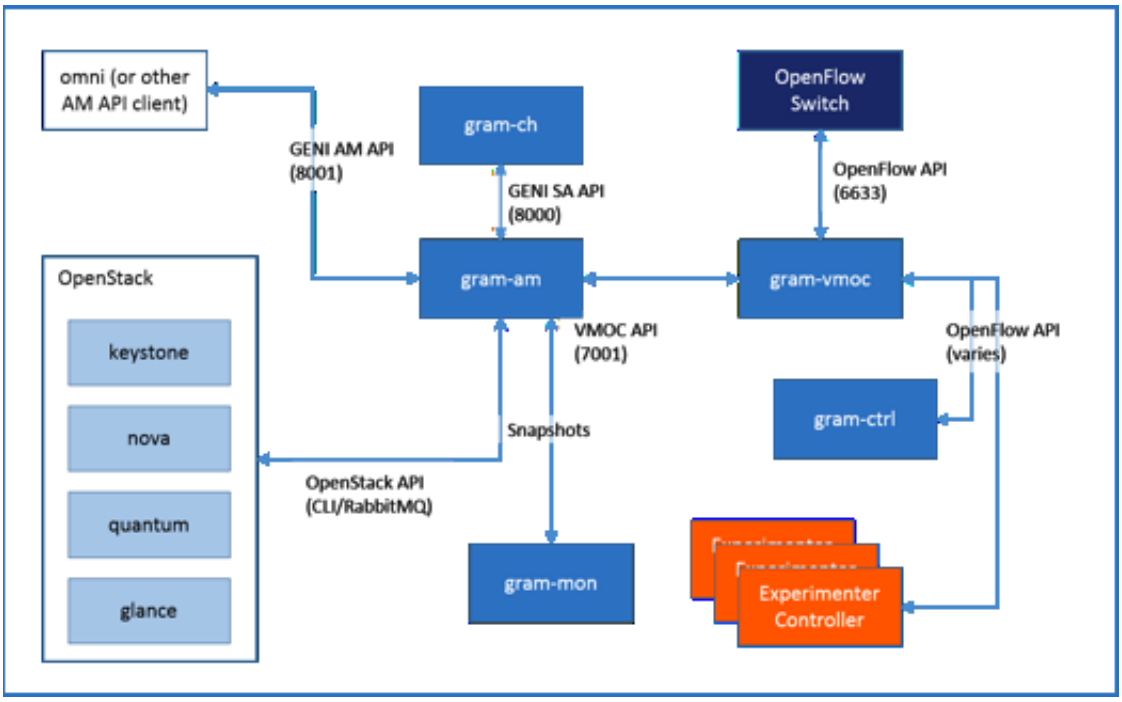


Figure 3.5: Omni Interface Architecture [26]

3.7 Fault Tolerance

As we all know, every technology comes with some demerits. Likewise, networking environments are dynamic, and the availability of the instances cannot be controlled. Although with time, mitigating strategies have taken place, it is hard to have 100% availability. SAGEFed cannot prevent these problems. The users are aware of such problems and the best way to deal with them is to predict the problem and take appropriate action.

3.8 Summary

In this chapter, we discussed the design of the principle components of SAGEFed and their functionality. Some design ideas in this chapter take a different view on how researchers deal with the federation of cloud computers. To cater similar func-

tionality on both SAVI and GENI instances, we have incorporated the most common functionality of acquiring, releasing and listing the resources. In the next chapter, we will see how SAGEFed takes advantage of the existing client tools and provides a better common system to process requests in both instances.

Chapter 4

Implementation

Your choices become your directions the moment you start to implement them. When you make choices and you dont implement them, your choices may be the best ever, but the most useless. – ***Israelmore Ayivor, Dream Big!***

4.1 Overview

In Chapter 3, we outlined the design of SAGEFed. In this chapter we take a deep dive and explore the implementation details of SAGEFed. Specifically, we will look into the implementation of version 1.0 of SAGEFed. Details of what can be done in version 2.0 is discussed in Chapter 6.

SAGEFed has been implemented using a shell script. The selection of the language was influenced by the working of the underlying backend client and the ease of modification in future. Also, the basic library provides all the functionality, requires no additional packages and avoids the updates of those libraries in future.

In this chapter, we will explain implementation with a different approach. First we will get familiar with a typical user's journey, and then we will discuss how it is implemented in SAGEFed. After we have discussed the implementation, we will

see how the existing users of SAVI will be migrated to the new LDAP (Lightweight Directory Access Protocol) server as discussed in Section 3.3

4.2 Use Case

User action is the course of steps that represents the intercommunication of various elements of the system to accomplish a purpose. Figure 4.2 pictures the journey where the user enters some command and how several components of SAGEFed interact to achieve the desired results. Once the users have the required credentials and set up, they can use SAGEFed and execute any available commands. After the command is executed, the validator first checks for the command that is the first argument. A sample command is shown below in Figure 4.1.

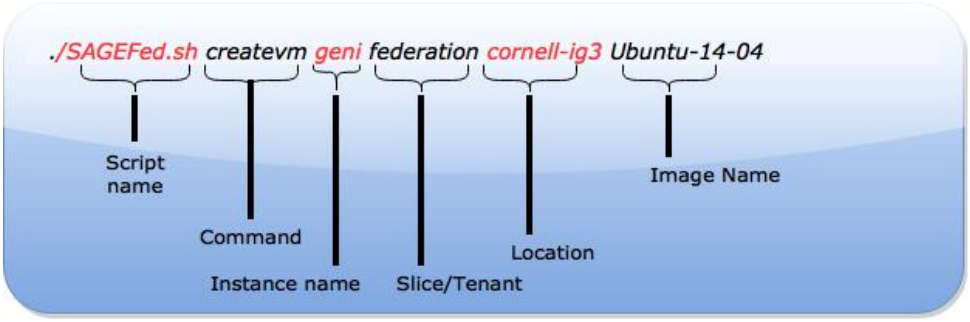


Figure 4.1: Sample Command

The script stores all the arguments in variables that can be used by the program at different stages. The first argument is the command. The validator examines the command and decides whether one of the provided commands is valid or not. If the command is valid, the validator checks the other supplied argument. Second argument determines the instance on which user wants to perform a certain action. The instance name is matched with the available two cases GENI and SAVI. In the event of any failure of validation, the script exits with an error message that is

shown as a display message in Figure 4.2. After all the validation is completed, the translator generates the understandable backend client command that is executed on the designated instance which can be either SAVI or GENI. The script then creates the logs, and the valuable information is displayed as a display message on the screen followed by the termination of the script. We will discuss the implementation of each component in the following sections.

4.3 SAGEFed

As mentioned earlier, SAGEFed is a wrapper around the client tools making a higher-level programing abstraction that sits above the SAVI and GENI clouds. We have made a comparison to neutralise the distinctive underlying architecture and terms supported by both the clouds. This comparison is important in discussing the working of the validator and translator. The structure of SAVI and GENI experiments were discussed in Sections 1.4 and 1.3. All the users are associated with restricted space on the cloud to maintain security. This restricted space is known as *Tenant-name* in SAVI and *Project* in GENI. Any user who has access to a particular Tenant-name or project can create and access the resources in that space only. GENI has one more layer of bifurcation which is known as *Slice*, see Section 1.3. In comparison, SAVI's tenant name is equivalent to a combination of Project and Slice. The location is the station where the cloud has their hardware and those can be used to reserve resources. As a result, the virtual machines are known as *instances* and *sliver* in SAVI and GENI respectively. Table 4.1 shows how the different terms are related to each other. It is important to note here that this mapping was only arrived upon after consulting with members of both the SAVI and GENI research communities.

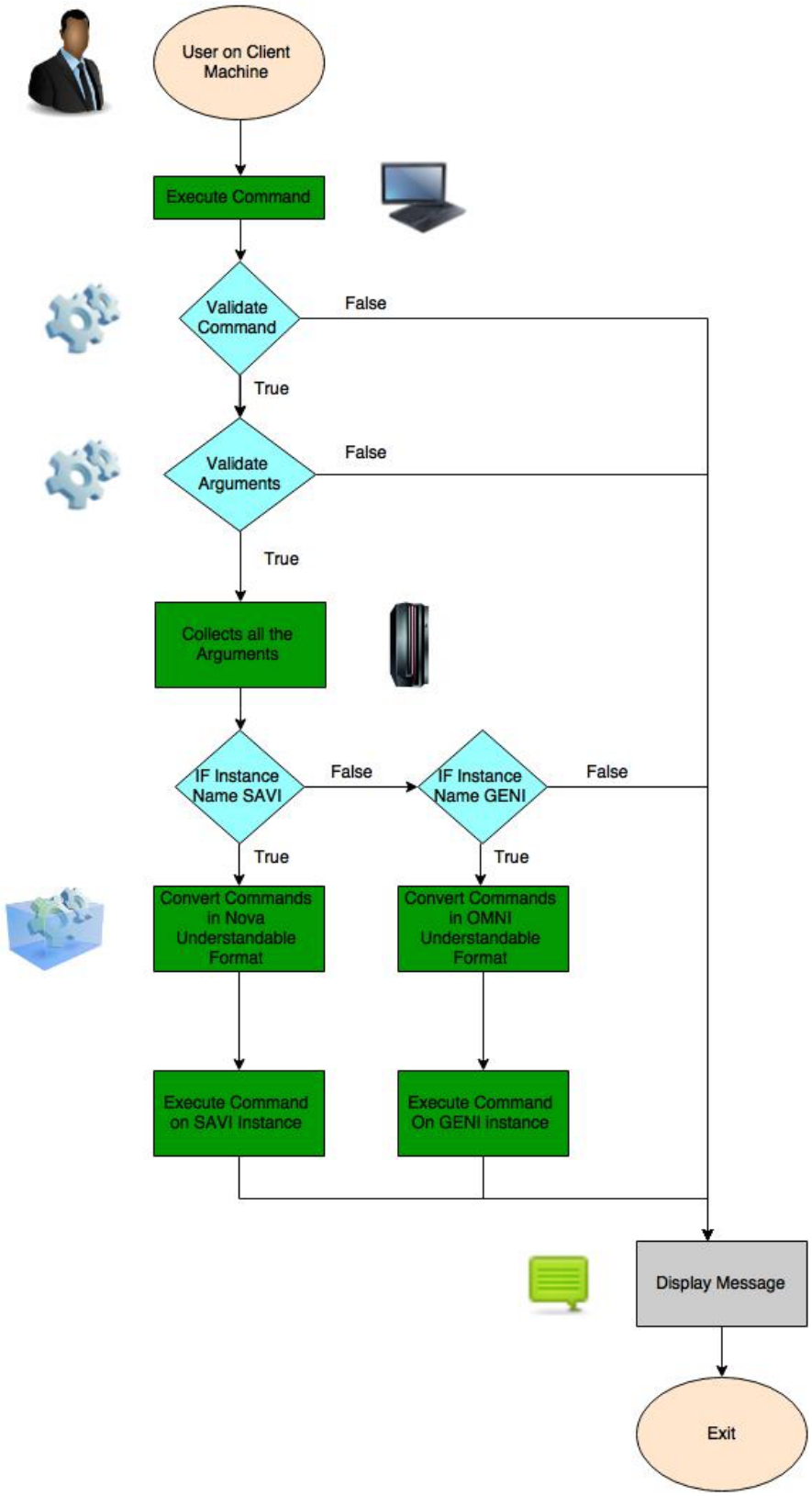


Figure 4.2: Use Case Flow Diagram

Terms	SAVI	GENI
Restricted User Space	Tenant-Name	Project and Slice
Operation System Images	Images	Rspecs
Location	Edge-Name	Rack-Name
Client Tool	NOVA	OMNI
Virtual Machine	Instance	Sliver

Table 4.1: Mapping Table

The folder structure of SAGEFed is clean and straightforward as shown in Figure 4.3. The image folder stores a file with the identifier names for all the operating systems' images supported by SAVI. Similarly, various RSpec files are required to produce a different type of virtual machine on GENI. All those different types of Rspec's files are stored in the RSpecs folder. When the script is executed, it creates several logs that are used to fetch useful information and display them for users. All the logs are stored in the logs folder with a timestamp attached to the name. The temporary folder stores all the intermediates logs that were created and used by SAGEFed.

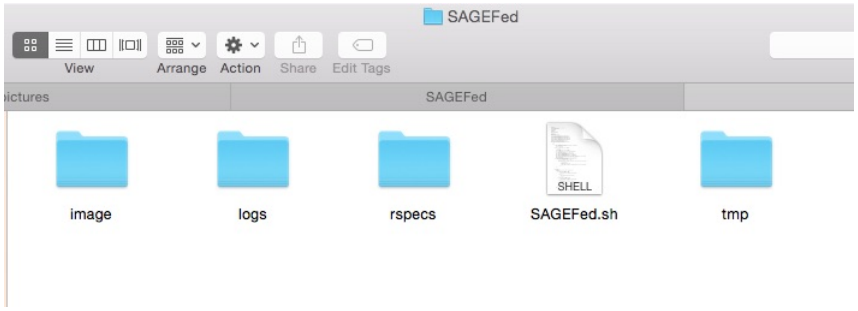


Figure 4.3: Folder Structure of SAGEFed

4.3.1 Validation

Validation is a three-step process. In the first step, the code validates the commands using a case statement. The second step usually includes the validation of the client

name. The third step is about finding whether all the required parameters have been supplied or not. Validation of some parameters supplied varies from command to command. Hence, all the commands are validated separately. SAGEFed expects a maximum of eight arguments. As all the arguments are stored in the variables, the check is made if the value of the last supplied argument is null or not and in some cases the value is tested for some specific value. If the value is null we exit the code or else we continue to call that function in the translator. For example, if a command is expected to have four arguments then after validating command and the client name SAGEFed examines the value in the fourth argument. Depending on the value of the fourth argument the execution progresses accordingly.

The command names are designed to be self-explanatory and reveal the purpose of those commands. The commands are discussed from the validation perspective.

1. ***createslice:*** This command creates the slice at GENI instance. As contended in the previous sections, this feature is available only for GENI and not for SAVI. Once the user executes the script with createslice as a command, the validator inspects for the client name. Post the client name validation the validator checks if the slice name is given or not and then takes the respective actions of either exiting the script or calling the translator function as shown in Figure 4.4.

2. ***deleteslice:*** To delete any slice at the GENI instance deleteslice command is facilitated with the client name and slice name. The validation is the same as the *createslice* command where the client name is validated first. The validator then looks for the slice name as an argument and check if the value provided is null or not as shown in Figure 4.5.

3. ***createvm:*** Createvm is used to create a particular type of virtual machine,

```
#create slice on GENI if validation pass
createslice)
    clear
    if [ "$client" = "SAVI" ]; then #checks if the client name is SAVI then exit as this feature is not valid for SAVI.
        echo "This feature is only available for GENI"
    elif [ "$client" = "GENI" ]; then #checks if the client name is GENI then go for the next step of validation.
       if [ -z "$slice_tenant_name" ]; then #validates if the slice name is provided or not.
            echo "Slice name not provided"
        else
            create_slice #call the translator function
       fi
    elif [ -z "$client" ]; then
        echo "Invalid Client Name."
    else
        echo "Invalid Arguments"
    fi
;;
```

Figure 4.4: Createslice Validation

```
#delete the slice from GENI
deleteslice)
    clear
    if [ "$client" = "SAVI" ]; then #check if the client name is SAVi then exit with appropriate message
        echo "This feature is only available for GENI"
    elif [ "$client" = "GENI" ]; then #validates the client name and take appropraite actions.<<<
        if [ -z "$slice_tenant_name" ]; then #validates if the slice name is provided or not.
            echo "Slice name not provided"
        else
            echo "Gathering Information..."
            delete_slice # call  the translator function
       fi

    elif [ -z "$client" ]; then
        echo "Invalid Client Name."
    else
        echo "Invalid Argument"
    fi
;;
```

Figure 4.5: Deleteslice Validation

which in turn used to run the experiments. It requires arguments such as tenant-name/slice-name, type of virtual machine, location where the instance needs to be created, etc. The validation process first studies the client name and then validates the value of the last argument supplied to SAGEFed script. In this case, the last argument is expected to be the VM name and is checked for the supplied value. If the validator is not able to find correct client name i.e. SAVI or GENI or the VM name is null, then it exits the script with proper error message. The code snippet is available in Figure 4.6

4. ***deletevm:*** As these research testbeds are used by the various organization, it is expected to release the resources one the experiment is completed. The deletevm command can be used to accomplish the task of releasing the resources. The

```
#create a new vm at different clients/servers
createvm)
    if [ "$client" = "SAVI" ]; then #checks if the client name is SAVI then exit as this feature is not valid for SAVI.
        if [ -z "$vm_name" ]; then #check if the last argument is null
            echo "Not all the arguments are provided"
        else
            clear
            echo "Creating VM on Savi Instance.."
            create_vm #call the translator function
        fi
    elif [ "$client" = "GENI" ]; then #checks if the client name is GENI then go for the next step of validation.
        if [ -z "$vm_type" ]; then #check if the last argument is null
            echo "Not all the arguments are provided"
        else
            echo "Creating VM on Geni Instance.."
            create_vm #call the translator function
        fi
    elif [ -z "$client" ]; then
        echo "No client Name Provided"
    else
        echo "Invalid Arguments"
    fi
;;
```

Figure 4.6: Createvm Validation

arguments supplied to release the resources are commands name (deletevm), client name(GENI, SAVI), slice name or tenant name. The VM name is supplied only in case of SAVI instance. The validation process checks for the supplied command name, client name and then validates the value of the last supplied argument as shown in Figure 4.7.

```
#delete the virtual machine and free the acquired resources
deletevm)
    clear
    if [ "$client" = "SAVI" ]; then #check if the client name is SAVi then exit with appropriate message
        if [ -z "$vm_type" ]; then #check if the last argument is null
            echo "Not all the arguments are provided"
        else
            echo "Deleting instance from Savi"
            delete_vm #call the translator function to take appropriate actions.
        fi
    elif [ "$client" = "GENI" ]; then #check if the client name is GENI then exit with appropriate message
        if [ -z "$slice_tenant_name" ]; then #check if the last argument is null
            echo "Not all the arguments are provided"
        else
            delete_vm #call the translator function to take appropriate actions.
        fi
    elif [ -z "$client" ]; then # if no client name is provided or is not SAVI or GENI  then exit the script
        echo "No client Name Provided"
    else
        echo "Invalid Arguments"
    fi
;;
#list all the resources reserved by the user
```

Figure 4.7: Deletevm Validation

5. ***listinstance:*** Once the virtual machine is created we can see the status, address and other necessary information through this command. Arguments required for the execution of this command are client-name and the slice/tenant

name. If both the parameters are provided, the list of all the virtual machines is displayed on the screen as shown in Figure 4.8.

```
#list all the resources reserved by the user
listinstance)
    clear
    if [ "$client" = "SAVI" -o "$client" = "GENI"]; then #validates the client name
        if [ -z "$slice_tenant_name" ]; then # check if the argument is passed or  not
          echo "Not all arguments are provided"
        else
          list_instance
        fi
    else
        echo "Not all arguments are provided."
    fi
;;
```

Figure 4.8: Listinstance Validation

4.3.2 Translation

Translation is a process of converting the user commands to backend client tools understandable format. Once the validator has done the basic validation, it calls the specific translator function. The first step in any translator function is to watch for the client name as the translator needs to process commands for both SAVI and GENI in a different way. After translator makes the first check, it starts transforming the commands. In the last step, the translator executes those converted commands, collects the logs and fetches the valuable message and display on the screen for the user. We will see each translator function in terms of a flow diagram to know understand how the system is built.

1. ***createslice:*** The creation of slice is associated just with the GENI instance and not SAVI. Hence, the validator makes sure to display the correct error message if a user supplies the client name as SAVI. Then the validator validates the command and sends it to the translator. The translator first analyzes the command executed by the user. The analyze phase is referred as, to store all

the arguments in different variables. After the analyze phase is over it identifies the correct client tool, i.e. Omni and converts the command into an Omni-understandable format. The log is stored in a file for future reference as shown in Figure 4.9. Finally, the translator fetches just the valuable information and displays that on the user screen

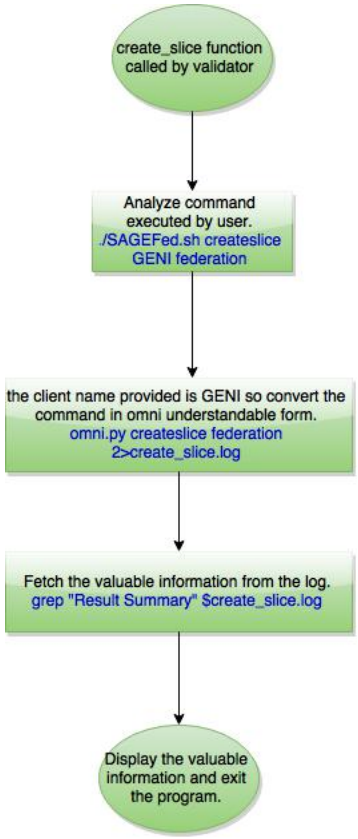


Figure 4.9: Createslice Translation Function

2. ***deleteslice:*** The deleteslice function is similar to createslice and is available for just GENI instance. Once the validator completes it initial validation, deleteslice function is called internally. The translator converts the command to Omni understandable format and executes it on GENI instance. Once the execution is completed, the useful information from the logs is fetched and displayed for

the user. Figure 4.10 shows the flow of the deleteslice translator function.

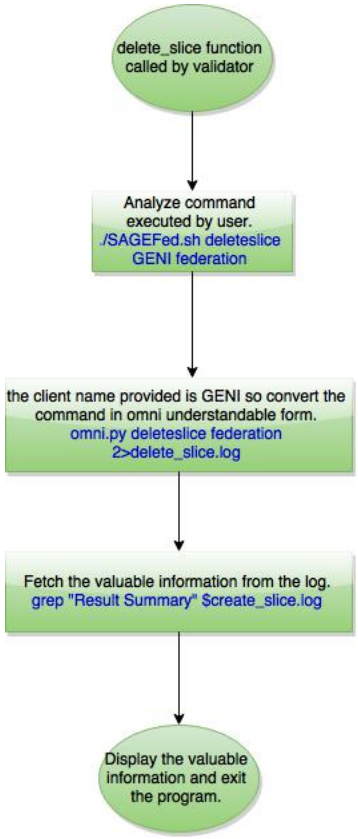


Figure 4.10: Deleteslice Translation Function

3. ***createvm:*** The createVM function is associated with both the clients and process; hence, the first step is to check the client name and progress accordingly. The whole journey is displayed in Figure 4.11. If the client name is GENI, then the first step is to look for the rspec file requested by the user in the rspec folder. If the file is available, then move to the next step and convert the user command and check for any failures. In the case of any failures, the error message is displayed, and the script quits. If there are no failures, it fetches the useful information and display it to the user. At last, all the logs are moved in logs folder with the date and time stamps appended to the name.

   The second case is when the client name supplied was SAVI. In this case the first step is to check for the supplied tenant name. The second step is to change the location of the specific edge name supported by SAVI and update the Nova configuration file with the new values. In the next couple of steps, the translator finds the image ID from cinder and selects the flavour ID for the relevant details provided by the user. After all the match-making is completed, the translator adds the supplied key pair to Nova and executes the command to create a virtual machine. To access the newly built virtual machine we need a public IP associated with the machine. Hence before the translator exits, it associates the IP address with the machine, displays the information on the screen and exits.

4. ***deletevm:*** Freeing up the reserved resources is a critical task in the world of cloud computing. As all these resources are shared among a considerable number of people, it is expected to release the resources as soon as the requirement is over. Every instance has its process to release the secured resources. In GENI, we cannot delete just one specific sliver (VM); rather, we have to eliminate the whole slice that in turn removes all the slivers on that particular slice. Once

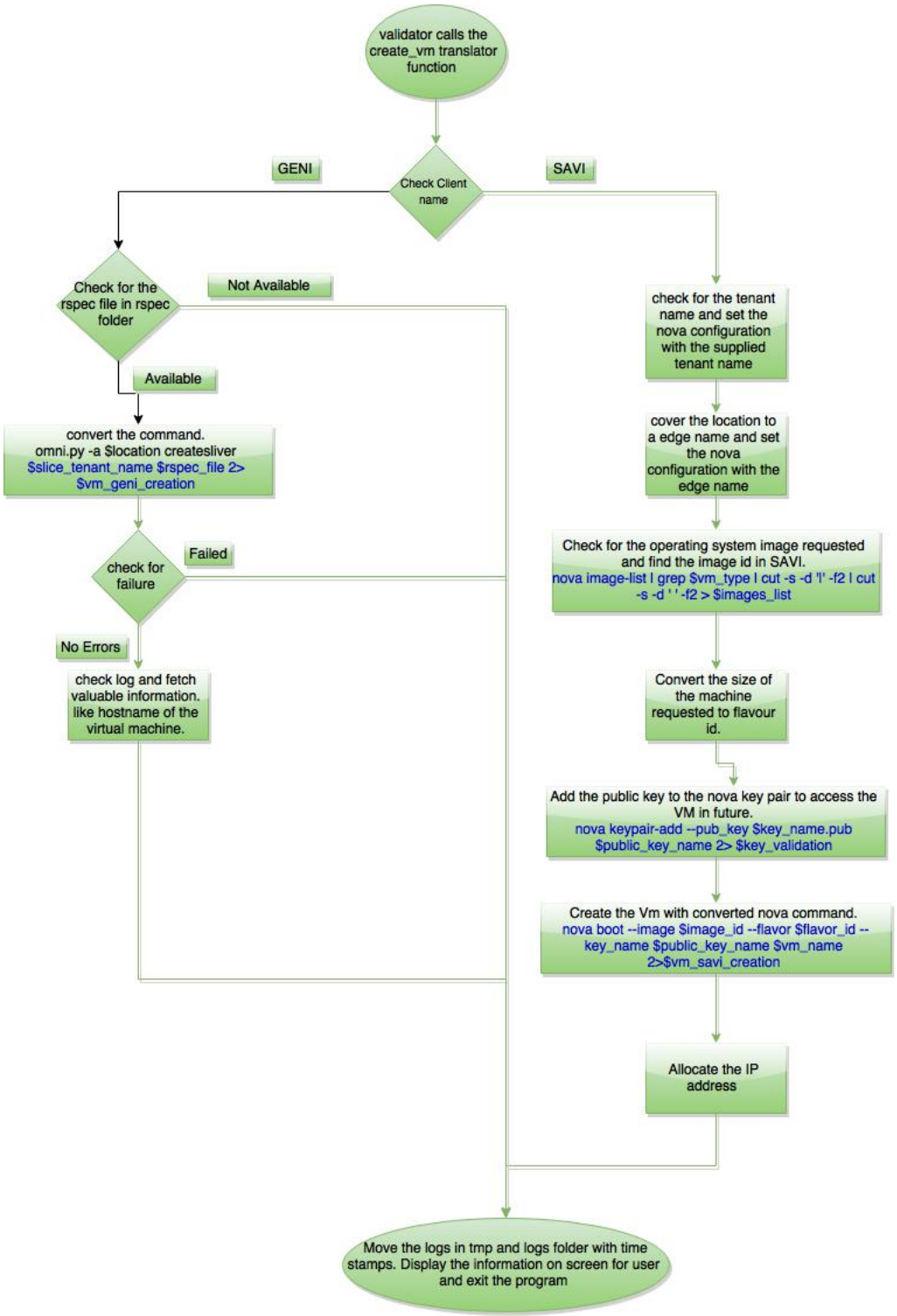


Figure 4.11: Create Virtual Machine Translation Function

the translator receives the command from the validator, it checks for the client name. For GENI, the first step is to look at whether the slice name is provided or not. If the slice name is not provided, it displays the error message and exits. Otherwise it converts the command and executes to release the resources.

If the client name is SAVI, the first and second steps are the same as in createvm. We fetch the tenant name and the location that is then converted into terms understandable by SAVI. The configuration file is updated and loaded. The third step is to execute the transformed command and look for errors. In the case of an error related to multiple instances, the user is asked to provide the exact instance id and re-run the command or else the appropriate success or error message is displayed on the screen for user reference.

5. ***listinstance:*** Listinstance is used to display all the instances available on the particular slice or edge. This command collects the values by executing different commands and combining them into a single output file. Once it collects all the information, it displays the full information on the screen. The information presented is the instance name, the hostname/IP-address and the status of the machine whether it is active or inactive. The command is useful to find the address of the machine that is used to connect to the virtual machine.

Command	No of attributes required in SAVI	No of attributes required in GENI
createslice	Not Applicable	One (slice name)
deleteslice	Not Applicable	One (slice name)
createvm	Six (location, tenant name, image name, machine size, public key name, machine name)	Three(location,slice name,rspec file)
deletevm	Two (location, vm name)	One (slice name)
listinstance	One (location)	One (slice name)

Table 4.2: Arguments required for each command

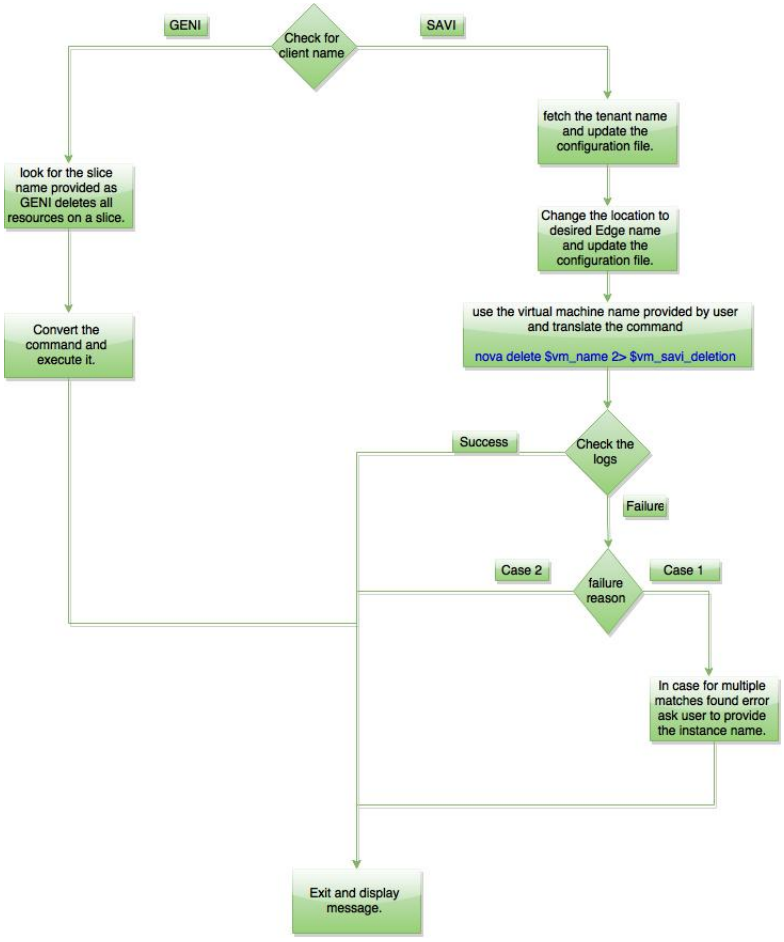


Figure 4.12: Delete Virtual Machine Translation Function

4.4 Migration of existing SAVI users

The SAGEFed is now ready and available for use. However, the challenge is how the existing users can use SAGEFed and the GENI resources. The solution to that problem is that, the users need to login to the GENI portal, and the GENI portal will direct the user to the newly set-up Shibboleth server. Then SAVI users should be able to use their existing user credentials to login to the Shibboleth server and in turn access the GENI portal. The shibboleth validates the credentials in the LDAP as discussed in Section 3.3. The real problem was how the existing encrypted credentials can be stored in LDAP without the user intervention.

SAVI uses the Python sha512 [61] to encrypt all passwords and stores them in the MySQL server. When the user tries to login on the SAVI portal, horizon sends the entered text password to Keystone to the sha512 library to match the supplied password with the stored encrypted password. The solution to the problem was very simple transfer of the user credentials to the LDAP in encrypted form. As the Unix-based system supports the sha512 algorithm, the LDAP server was built on the Unix-based system. The only difference is the Unix-based system needs to know which encryption is needed to decrypt the password.

The migration of existing users is the one-time activity that was performed to migrate all the user credentials in encrypted form. The script converted the password in the Unix-based understandable format and then saved to the LDAP server. All the credentials were imported in an SQL format file from Keystone MySQL database. The SQL file is used to create the LDAP readable format as shown in Figure 4.13. Once the migration process was completed, all existing users were able to access the Shibboleth server with their existing username and password.

```
dn: uid=mdarianian,dc=ldap,dc=savitestbed,dc=ca
uid: mdarianian
cn: mdarianian
sn: mdarianian
mail: mohamad.drnn@gmail.com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
objectClass: shadowAccount
userPassword: {crypt}$6$r3GmJvZIYQ6dg$aIDMUyTtXMcNGd/veGdgl8CF5Z7/0yU0IiRMVVW1WXYDvGIAFZu2TaLAu7cBFXywtEg6kB9cirTuq54q2lmLv0
shadowLastChange: 16552
shadowMin: 0
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
uidNumber: 1000
gidNumber: 1000
homeDirectory: /home/mdarianian
gecos: mdarianian
```

Figure 4.13: Sample LDAP Format File

4.4.1 Adding New SAVI users to LDAP

Research beds such as GENI and SAVI are used all over the world. Hence, the addition of new users and the modification of user details are very common cases. Whenever a new user demands access to SAVI, the admin approves the request and builds an account for the user. The user entry is created in the OpenStack MySQl database. These login credentials need to be added to the LDAP server as well. Figure 4.14 is the script that can be used by the admin to add these users to LDAP. The script was first created to add any users manually but then was changed to run in a cronjob so that the details are saved automatically. The script essentially requires three arguments: user ID, email address and encrypted password. Once all three arguments are passed, the script creates an LDIF file (LDAP understandable format) with other appropriate information and executes the LDAP command to save it to the LDAP server. This script is run on a daily basis to sync the Keystone database with the LDAP database.

```
#/bin/bash

#This script will add the new users to the SAVI LDAP server.

file="/home/savi/add_new_user.ldif"
[[ -f "$file" ]] && rm "$file"

touch "$file"
ls "$file"

id=$1
email=$2
password=$3

#echo -n "Please Enter the USER ID: "
#   read id
#echo -n "Please Enter the EMAIL ID: "
#   read email
#echo -n "Please Enter new Password (encrypted value): "
#   read password

echo $id
echo $email
echo $password

echo "dn: uid=$id,dc=ldap,dc=savitestbed,dc=ca" > "$file"
echo "uid: $id" >> "$file"
echo "cn: $id" >> "$file"
echo "sn: $id" >> "$file"
echo "mail: $email" >> "$file"
echo "objectClass: person" >> "$file"
echo "objectClass: organizationalPerson" >> "$file"
echo "objectClass: inetOrgPerson" >> "$file"
echo "objectClass: posixAccount" >> "$file"
echo "objectClass: top" >> "$file"
echo "objectClass: shadowAccount" >> "$file"
echo "userPassword: {crypt}$password" >> "$file"
echo "shadowLastChange: 16552" >> "$file"
echo "shadowMin: 0" >> "$file"
echo "shadowMax: 99999" >> "$file"
echo "shadowWarning: 7" >> "$file"
echo "loginShell: /bin/bash" >> "$file"
echo "uidNumber: 1000" >> "$file"
echo "gidNumber: 1000" >> "$file"
echo "homeDirectory: /home/$id" >> "$file"
echo "gecos: $id" >> "$file"


ldapadd -x -h 127.0.0.1 -D "cn=admin,dc=ldap,dc=savitestbed,dc=ca" -w uvic101 -f "$file"
```

Figure 4.14: Add User Code Snippet

4.4.2 Modify or Delete the existing user details

Modification and deletion are the other two commonly occurring scenarios. As GENI has configured and believe that, the certificates are sent by Shibboleth, so it is necessary to keep the LDAP server up-to-date. The modification request is usually generated for either change in email address or password. Hence, the modification script provides these two options and expects the admin to provide the necessary information. Figure 4.15 is the code snippet for the same. On the other hand, deletion

is far simpler and just needs the user ID that then removes the entry from the LDAP server for that particular user.

```
echo " 1. Change User Password."
echo " 2. Change User Email-id"
echo -n " Please enter your choice: "
read choice
if [ $choice == 1 ]; then
    clear
    echo -n "Please Enter the USER ID: "
    read ID
    echo -n "Please Enter new Password (encrypted value): "
    read PASSWORD
    file="modify_user_password.ldif"
        [[ -f "$file" ]] && rm "$file"
    echo "dn: uid=$ID,dc=ldap,dc=savitestbed,dc=ca" >> modify_user_password.ldif
    echo "changetype: modify" >> modify_user_password.ldif
    echo "replace: userPassword" >> modify_user_password.ldif
    echo "userPassword: {crypt}$PASSWORD" >> modify_user_password.ldif
    ldapmodify -x -h 127.0.0.1 -D "cn=admin,dc=ldap,dc=savitestbed,dc=ca" -w uvic101 -f modify_user_password.ldif


else
    echo $choice
    clear
        echo -n " Please Enter the USER ID: "
        read ID
        echo -n "Please Enter new EMAIL-ID: "
        read EMAIL
    file="modify_user_email.ldif"
        [[ -f "$file" ]] && rm "$file"
    echo "dn: uid=$ID,dc=ldap,dc=savitestbed,dc=ca" >> modify_user_email.ldif
    echo "changetype: modify" >> modify_user_email.ldif
    echo "replace: mail" >> modify_user_email.ldif
    echo "mail: $EMAIL" >> modify_user_email.ldif
    ldapmodify -x -h 127.0.0.1 -D "cn=admin,dc=ldap,dc=savitestbed,dc=ca" -w uvic101 -f modify_user_email.ldif
fi
```

Figure 4.15: Modify user code snippet

4.5 How to use SAGEFed

Any user who wants to access the resources on both GENI and SAVI and wishes to avoid learning the different commands to achieve this goal can use SAGEFed. As presented in A.1 the user first needs to follow the steps to get a user account in different instances. The user can then download the shell script and upload the Omni bundle to the SAVI client machine and start using it immediately. The SAVI client machine has the preinstalled updated Nova client version that helps to avoid any installation. In case users want to set up SAGEFed on their personal computer, they can use the setup script. The configuration script guides the users to set up both Nova and Omni on their personal computer. Once the installation is complete, the user can use the command line to execute various available commands in SAGEFed. When the script is run for the first time, the Nova configuration settings are done for which users need to enter their credentials. By default, the user is logged on the Toronto edge but they can adopt any edge while performing the commands. Also, all the logs are collected in the logs folder so users can easily check the logs if they need to find more out about an execution or error.

4.6 Summary

This chapter discussed intricate implementation details of numerous components in SAGEFed. The implementation, through the use of shell script, is made easy for any eager developers to pick up and contribute. Alongside with the very detailed implementation discussions, this chapter presents a straight forward work flow for normal users.The work flow is kept very simple, so users can adopt SAGEFed easily. In the next chapter we will see the experiments and their results with the goal of verifying the validity of a problem.

Chapter 5

Evaluation

*Take a step back, evaluate what is important, and enjoy life.–**Teri Garr***

The evaluation of SAGEFed is based upon the performance and usability of the application. SAGEFed is a tool that uses the available client tools to perform common actions around cloud. Our assessment is to compare the time taken to complete a task by SAGEFed with the times of the client tools. The other assessment factors are the usability study that measures the application's potential to achieve the goal of reserving the resources and managing them in a way that has less cognitive burden on the users than the current approach, which requires familiarity with two different APIs and two different sets of terms for the resources involved. Below are the factors that are involved in the usability study. We use an exploratory mixed-methods approach, with both quantitative and qualitative metrics.

1. Efficiency, as measured by time taken by the users and comparing the number of steps executed by user to achieve the goal.

2. User Experience, concluded on the basis on users feedback. The feedback was focused on how users feels, reacts while using the application and their view on the use of SAGEFed

5.1 Efficiency

The efficiency is based on various parameters taken into account. The first parameter is the number of commands executed by the user to perform a function. A comparison can be done with the number of commands that need to be executed while performing the same task with the client tool and SAGEFed. Roughly speaking, fewer commands to engage a task means less cognitive overhead for the user. Another parameter is the information provided by the logs. The information in the logs generated by the client tools, consist of some information which is not essential for most of the users. Hence, the logs were cleaned to display just the useul information.

While using the client tools, the user has to run many commands to achieve a simple goal, for example, creating a virtual machine. In this case, the user might need to remember the different arguments required by several commands to achieve an objective.Our application provides a common API and performs all the steps required to reach the goal automatically. As a result, the SAGEFed reduces the number of manual steps to be performed and saves the user overhead involved.

5.2 User Experience

User experience is all about how the person feels, reacts and views the use of an application. It is a very important factor, as it provides the qualitative analysis and helps to improve the way users interact with the system. As it is admittedly hard to measure this parameter, it is usually measured in satisfaction value. Is the user satisfied or dissatisfied with the usage of an application? A scale of 1 to 10 measures the degree of satisfaction in the user feedback, which can be used to improve features associated directly with the user experience.

5.3 Experiment

An experiment is an arranged procedure conducted with the goal of answering, verifying the validity of a problem. For our problem, we conducted the same experiment with different types of people/groups as listed below. The experiment is available in Appendix A.2.

1. The experienced user
2. The novice user
3. The big groups

5.4 Experiment 1: Experienced User

We performed both experiment 1 and 2 using the tutorials referred to in Appendix A.2. The experiment is performed by two participants and both participants executed the tutorials as referred to in A.2. All the experiments were carried out on the SAVI client machine to avoid any installation of the client tools.

In the Experiment 1, the participant is an experienced user and has some familarity using the client tools. Participant A is first asked to complete the user journey with the client tool and then SAGEFed. In this process, we measure the timing of each stage, and the number of steps followed by the user. The different phases are:-

1. Creating a VM on SAVI & GENI both public IP.
2. List all the instances available on that Edge with the status & address of the machine.
3. Delete the VM created on SAVI & GENI.

5.4.1 Evaluation by Participant A

Efficiency

Operation on SAVI

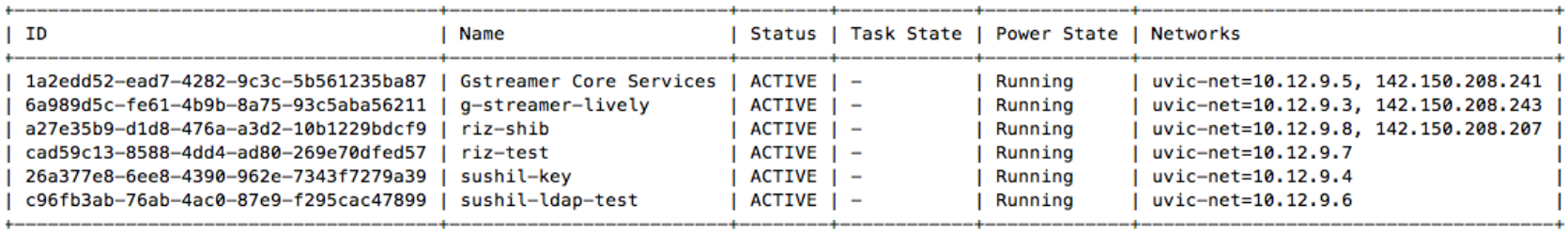
While creating a VM on SAVI using the client tools, the user first has to configure the configuration file. After the configuration is done, the user needs to find the OS image ID, flavour ID that is the machine (small, tiny, medium, etc.). Once the participant has all the desired information, the user has to add the key pair to SAVI to access the VM later on. Finally, the VM creation command is executed. Once the machine is created, a public IP can be associated to that machine and to do so, the user has to perform three more steps. Listing the instance is just a two-step process and it displays the information shown in Figure 5.1. Finally, the deletion of VM requires the user to find the instance ID and then the machine can be deleted from the SAVI server.

Next, the user was asked to perform the same steps using SAGEFed. SAGEFed achieves the same goal of reserving and releasing the resources with a single command. Hence, it provides liberty to the users to avoid executing multiple subcommands to perform a simple task. The steps such as finding the OS image ID, flavour id (small, tiny, medium), assigning the public key and providing the public IP address these are controlled by SAGEFed internally. Figure 5.2 displays the information posted by SAGEFed, which is similar to what we have seen in Figure 5.1. The only difference is that only valuable information is displayed on the screen. The time is the combination of the time taken by the user to execute the commands and the time taken by the process to perform that action. Table 5.1 presents the comparison of steps involved and the time taken by each participant to perform the task on the SAVI instance.

The value of the time is rounded off for the ease of comparison.

Stage	Using NOVA		Using SAGEFed Tool	
	Steps	Time(seconds)	Steps	Time(seconds)
Creating a VM	7	180	1	120
List Instances	2	30	1	25
deleting a VM	2	30	1	25

Table 5.1: Experiment 1 Comparison for SAVI



ID	Name	Status	Task State	Power State	Networks
1a2edd52-ead7-4282-9c3c-5b561235ba87	Gstreamer Core Services	ACTIVE	-	Running	uvic-net=10.12.9.5, 142.150.208.241
6a989d5c-fe61-4b9b-8a75-93c5aba56211	g-streamer-lively	ACTIVE	-	Running	uvic-net=10.12.9.3, 142.150.208.243
a27e35b9-d1d8-476a-a3d2-10b1229bdcf9	riz-shib	ACTIVE	-	Running	uvic-net=10.12.9.8, 142.150.208.207
cad59c13-8588-4dd4-ad80-269e70dfed57	riz-test	ACTIVE	-	Running	uvic-net=10.12.9.7
26a377e8-6ee8-4390-962e-7343f7279a39	sushil-key	ACTIVE	-	Running	uvic-net=10.12.9.4
c96fb3ab-76ab-4ac0-87e9-f295cac47899	sushil-ldap-test	ACTIVE	-	Running	uvic-net=10.12.9.6

Figure 5.1: List of Resources Using Client Tool

```
CORE
Gathering Information from Savi Servers...
+-------------------------------------------------------------------------------------------------------------+

| Instance_Name                                | Current_Status | Owner | Address                                    |
+----------------------------------------------+----------------+-------+--------------------------------------------+

| g-streamer-lively                            | ACTIVE         | Savi  | uvic-net=10.12.9.3,142.150.208.243         |
| riz-shib                                     | ACTIVE         | Savi  | uvic-net=10.12.9.8,142.150.208.207         |
| riz-test                                     | ACTIVE         | Savi  | uvic-net=10.12.9.7                         |
| sushil-key                                   | ACTIVE         | Savi  | uvic-net=10.12.9.4                         |
| sushil-ldap-test                             | ACTIVE         | Savi  | uvic-net=10.12.9.6                         |
+-------------------------------------------------------------------------------------------------------------+
sushilb@savi-client1:~/tutorial$ nova list
```

Figure 5.2: List of Resources Using SAGEFed

Operation on GENI

Using Omni is quite straightforward as compared to Nova. All the required configurations are stored in Omni bundle. Also, the Omni bundle holds the credential information of the user related to a specific project. Omni was designed to lessen the total steps required to perform a several task, for example, creating a slice and VM. The only shortcoming of using Omni is the information available in the logs. The valuable information is accompanied by the noise that makes it hard for a user to understand the output of the process, as shown in Figure 5.3. SAGEFed separates the noise and displays only the relevant information to the user as shown in Figure 5.4. Also, the user is required to execute several commands to find the details such as hostname, IP address and port number of the VM. SAGEFed's *listinstance* command is built to provide such information to the user. Figure 5.5 display the output of the *listinstance* command when executed on a GENI instance. Table 5.2 presents the results of Experiment 1 performed on the GENI instance.

Stage	Using OMNI		Using SAGEFed Tool	
	Steps	Time(seconds)	Steps	Time(seconds)
Creating a VM	1	60	1	60
List Instances	5	180	1	130
Deleting a VM	1	60	1	60

Table 5.2: Experiment 1 Comparison for GENI

```
15:17:31 INFO    :   <rspec expires="2015-08-16T19:09:04Z" type="manifest" xmlns="http://www.geni.net/resources/rspec/3" xmlns:emulab="http://www.protogeni.net/reso
urces/rspec/ext/emulab/1" xmlns:jacks="http://www.protogeni.net/resources/rspec/ext/jacks/1" xmlns:tour="http://www.protogeni.net/resources/rspec/ext/apt-tour/1" xm
lns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.geni.net/resources/rspec/3 http://www.geni.net/resources/rspec/3/manifest.xsd">
  <node client_id="node-0" component_id="urn:publicid:IDN+geni.it.cornell.edu+node+pc2" component_manager_id="urn:publicid:IDN+geni.it.cornell.edu+authority+cm" exc
lusive="false" sliver_id="urn:publicid:IDN+geni.it.cornell.edu+sliver+20378" xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1">
    <icon url="https://portal.geni.net/images/Xen-VM.svg" xmlns="http://www.protogeni.net/resources/rspec/ext/jacks/1"/>
    <site id="128" xmlns="http://www.protogeni.net/resources/rspec/ext/jacks/1"/>
    <routable_control_ip xmlns="http://www.protogeni.net/resources/rspec/ext/emulab/1"/>
    <sliver_type name="emulab-xen">
      <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops:UBUNTU14-64-STD"/>
    </sliver_type>
    <services>      <login authentication="ssh-keys" hostname="pcvm2-2.geni.it.cornell.edu" port="22" username="jbourne"/>      <login authentication="ssh-keys" hos
tname="pcvm2-2.geni.it.cornell.edu" port="22" username="andib"/>      <login authentication="ssh-keys" hostname="pcvm2-2.geni.it.cornell.edu" port="22" username="st
redger"/>      <login authentication="ssh-keys" hostname="pcvm2-2.geni.it.cornell.edu" port="22" username="acb"/>      <login authentication="ssh-keys" hostname="pc
vm2-2.geni.it.cornell.edu" port="22" username="nriga"/>      <login authentication="ssh-keys" hostname="pcvm2-2.geni.it.cornell.edu" port="22" username="jcappos"/>
     <login authentication="ssh-keys" hostname="pcvm2-2.geni.it.cornell.edu" port="22" username="rickmcg"/>      <login authentication="ssh-keys" hostname="pcvm2-2.
geni.it.cornell.edu" port="22" username="jmambre"/>      <login authentication="ssh-keys" hostname="pcvm2-2.geni.it.cornell.edu" port="22" username="jimchen"/>
 <login authentication="ssh-keys" hostname="pcvm2-2.geni.it.cornell.edu" port="22" username="sushilb"/>      <emulab:console server="pcvm2-2.geni.it.cornell.edu"/>
   </services>
    <emulab:vnode name="pcvm2-2"/>    <host name="node-0.sushilomni.ch-geni-net.geni.it.cornell.edu"/>  </node>
  <rs:site_info xmlns:rs="http://www.protogeni.net/resources/rspec/ext/site-info/1">    <rs:location country="US" latitude="42.453877" longitude="-76.484370"/>  </r
s:site_info></rspec>
15:17:31 INFO    : Reservation at cornell-ig in slice sushilomni expires at 2015-08-16 19:09:04 (UTC).
15:17:31 INFO    :  ------------------------------------------------------
15:17:31 INFO    :  Completed createsliver:

  Options as run:
               aggregate: ['cornell-ig3']
               framework: portal
               project: GeniExperimentEngine

  Args: createsliver sushilomni Ubuntu-14-04.rspec

  Result Summary: Got Reserved resources RSpec from geni-it-cornell-edu. Reservation at cornell-ig in slice sushilomni expires at 2015-08-16 19:09:04 (UTC).
15:17:31 INFO    :  ======================================================
```

Figure 5.3: Additional Information With Omni Commands

```
Configuring Enviornmental variables. Please Wait..
Creating VM on Geni Instance..
/home/savitb/sushilb/tutorial/logs/vm_create_geni.log


  Result Summary: Got Reserved resources RSpec from geni-it-cornell-edu. Reservation at cornell-ig in slice sushil expires at 2015-08-16 19:08:31 (UTC).
"hostname="pcvm2-3.geni.it.cornell.edu"

 To connect to the created VM please use to the given hostname="pcvm2-3.geni.it.cornell.edu"
```

Figure 5.4: Only Valuable Information

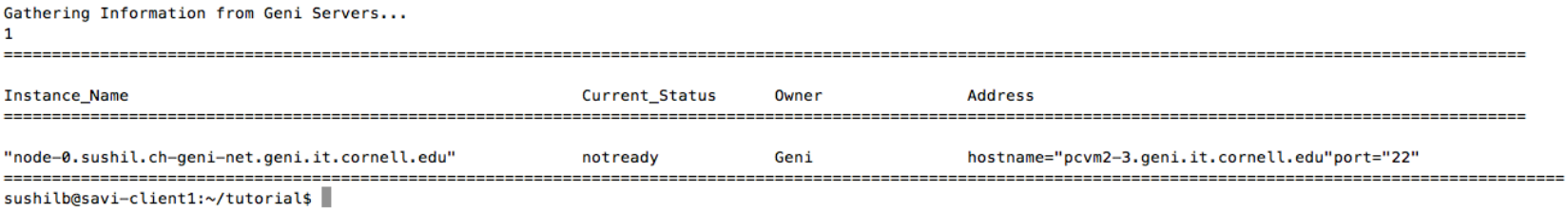


Figure 5.5: List of Resources Using SAGEFed

User's Experience: Satisfied with the performance of the application.

At the completion of the experiment, Participant A was asked for his verdict about our application. He was satisfied with the experience and will use the SAGEFed in future rather than the client tools. The user was made aware of the limited basic functionality provided by the application. Also, he was happy with the fact that he could run the client tools commands too.

He found it useful as he does not have to remember multiple commands. Also, location name can be supplied in the form of a city name for SAVI, and for GENI an aggregate name can be provided rather than a URL. Additionally, the application was not adding any overhead to the performance but certainly improved the user experience with reduced number of steps and improved logs quality.

He wanted to have more information displayed on the screen regarding which logs should be referred for a particular operation. In future work, the application will present which precise logs a user should check to gather more information. As of now, the user can go to the logs folder, identify the log's name and review them to gather more information.

5.5 Experiment 2: Novice User

In the second experiment Participant B was a new user and had no experience working with any of the client tools. She was asked to use both tutorials and perform the actions similar to Experiment 1. This experiment is important because there will be so many new users in future who will use SAGEFed and such experiments can provide a handy comparison between the client tools and SAGEFed usability.

5.5.1 Evaluation by Participant B

Efficiency

Operation on SAVI

As the user is new and has no experience using any of the client tools, unlike Participant A the user had difficulty in task execution. The main point that needs to be noted here is the execution time was the same but the time taken by her to type those commands with the appropriate arguments was high as compared to Experiment 1. Table 5.3 represents the time taken and the number of steps taken by a user to complete the specified milestones.

Stage	Using SAVI		Using SAGEFed Tool	
	Steps	Time(seconds)	Steps	Time(seconds)
Creating a VM	7	200	1	120
List Instances	2	60	1	25
Deleting a VM	2	60	1	25

Table 5.3: Experiment 2 Comparison for SAVI

Operation on GENI

Again in this part of the experiment the user was quick as the number of steps involved were fewer but the additional time was due to unfamiliarity with the new commands. Table 5.4 display the comparison.

Stage	Using OMNI		Using SAGEFed Tool	
	Steps	Time(seconds)	Steps	Time(seconds)
Creating a VM	1	70	1	60
List Instances	5	200	1	130
deleting a VM	1	60	1	60

Table 5.4: Experiment 2 Comparison for GENI

User's Experience: Highly satisfied with the usability of the application.

In the second experiment, Participant B was highly satisfied with the new application. She was not aware of the client tools and was using them for the first time.

Hence, she preferred SAGEFed over the client tool because she had to execute fewer steps. Also, she was happy to have common commands and felt it was smooth to use the commands that resemble colloquial language. She was pleased to see that people who are familiar with the client tools can still use the system they want and switch to the new application with no additional changes.

5.6 Experiment 3: Deploy at Scale within an Application

The third experiment was to test the tool on a large scale. This tool was presented in three major conferences GEC 23 [1], 10th EAI International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities [62] and SAVI AGM [22]. The tutorial in A.2 was executed by over 100 users. The motivation behind this experiment was to test the SAGEFed capability of handling multiple users at the same time. Although no formal feedback was recorded, the result was 100% completion rate by all participants in each event, with no errors while executing the tutorial.

Conference Name	User Background	No. of Users
GEC-23	Researchers and Experienced Users	20
Trident Education Conference-2015	Researchers, Experienced and Novice Users	40
SAVI AGM	Students, Experienced and Novice Users	100

Table 5.5: Experiment 3 users details

5.7 Impact of federation over Applications

Ignite Distributed Collaborative Scientific Visualization System is a system designed to permit real-time interaction and visual collaboration around large data sets, with an initial emphasis on scientific data. The Ignite Visualization System offers such a collaborative environment, with real-time interaction on any device between users separated across a wide area. The Ignite Visualization System provides seamless interaction and immediate updates even under heavy loads and when users are widely separated. The design goal was to fetch a data set consisting of 30,000 points from a server and render it within 150 milliseconds, for a user anywhere in the world, and reect changes made by a user in one location to all other users within a bound provided by network latency. The system was demonstrated successfully on a significant worldwide air pollution data set, with pollution values on a 10km, 25km, 50km, and 100km worldwide grid, with monthly values over an 18-year period. It was demonstrated on a broad range of clients, including laptop, tablet, and Smartphone. 5.6 displays the main screen of the application.

In such applications, it is important to have the server near to the user; the developer should also have access to all the servers. The application needs to be distributed over various servers all over the world which can be possible with the federation. Table 5.6 presents the comparison of a case where the application is available on just one server, and people are trying to access the same from different parts of the world. Also, if the same operation is performed with an application available on various servers all over the world and users are accessing the web server located closer to their physical location. The determining factor here is the time taken by the web server to retrieve the information and display it on the user screen. Such an application presents the need for federation and tools like SAGEFed, which provides the federation among different clouds using same commands and credentials.

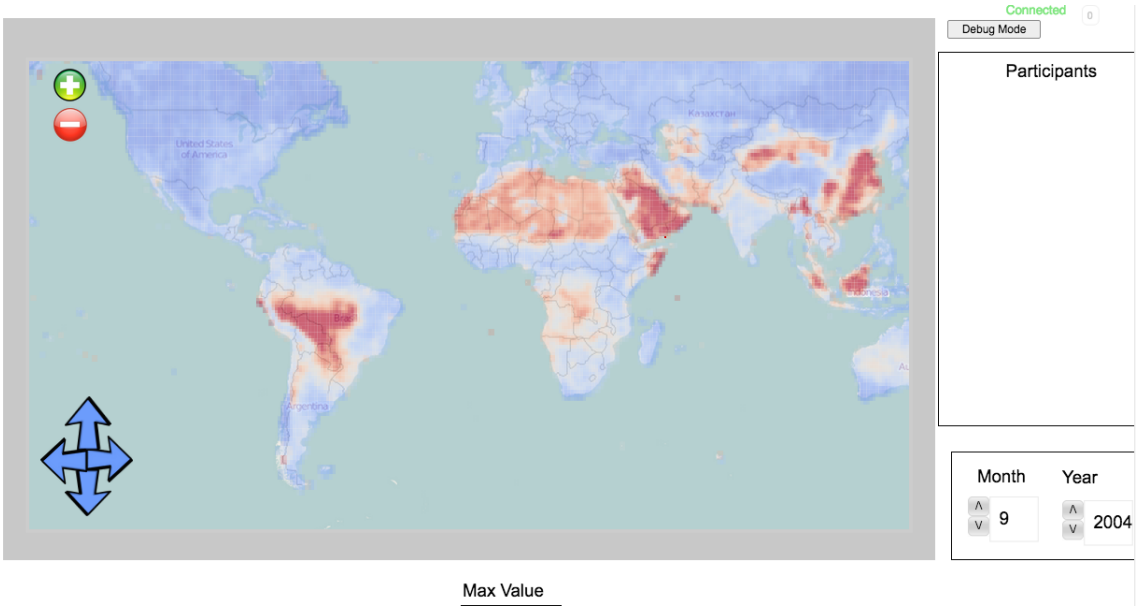


Figure 5.6: Pollution Visualizer

Location	User Victoria	User Japan	User America
Victoria	8	12	20
Japan	20	18	8
America	12	18	8

Table 5.6: Time taken in Milliseconds to retrieve information

5.8 Summary

In this chapter, we evaluated SAGEFed application with available client tools, i.e., Nova and Omni. The applications were assessed with the help of 160 users. These users completed a whole journey with both SAGEFed and available client tools and provided their feedback. The usability of an application was assessed on the basis of efficiency and user experience. Participant A, who had some experience using these client tools, was satisfied with the new application and suggested few modifications. Participant B, who had never used any such application or client tool, was highly satisfied with the application. She even recommended SAGEFed over client tools. Both participants were happy with the approach of using a common API and the

way SAGEFed helps to reduce the execution steps.

Chapter 6

Conclusions

I think and think for months and years. ninety-nine times, the conclusion is false. The hundredth time I am right. – ***Albert Einstein***

The intention of developing SAGEFed was to provide a proof of concept that interoperability in federated clouds is possible without going through major architectural changes. In this thesis we were able to show that SAGEFed was designed in such a way that it caused little overhead and provided basic human readable commands. It not only provided users with ease of use, but also provided a common API to perform various functions over the two different clouds. SAGEFed is a shell script that is lightweight, as it takes advantage of the available client tools. The shell script is available for users with simple commands like *createvm, deletevm, and listinstance.* SAGEFed automatically translates these commands to any client tool understandable format. As a wrapper around the client tools, SAGEFed can interact with those client tools for the user. The different arguments help SAGEFed to translate the executed command and perform the desired function on the respective cloud instances.We have been able to show that 160 users were able to seamlessly execute on both GENI and SAVI in our experiments so far.

6.1 Future Work

As we discussed in Chapter 5, SAGEFed can be used for resource allocation for heterogeneous applications. However, there are many features that can be implemented to improve SAGEFed. At this point, SAGEFed just supports interoperability between GENI and SAVI but small modifications can be done to make it work with other clouds such as Compute Canada. The SAVI implementation can be used to perform actions on both Compute Canada and SAVI as they both are mere replicas of OpenStack architecture. This way we can include more resources and provide the benefits of resources to the users all over Canada and the USA. The downside of using such approach may make the SAGEFed script bulky in size.

The portals provide an excellent graphical user interface but lack the view of interoperability and ability to use the same API. In future, GRAM, a common API tool created by GENI, can be modified to work with the latest versions of OpenStack, which would solve the problem of the common API. Also, more features such as Live migration and load balancing can be implemented. As a solution to the common resource crunch problem, where all the resources on a cloud are reserved, and space is exhausted for new instances. The admin user can create a snapshot of the current state of the virtual machine and then migrate that machine to another cloud or edge. SAGEFed being a command line tool will make it easy to implement the automatic migration process.

Finally, since SAGEFed is acting as a wrapper over the client tool, accommodating more and more client tools can federate more clouds. The common API can be used to allocate resources anywhere in the world, which in turn will help the applications discussed in Chapter 5.

6.2 Final Thoughts

The SAGEFed tool covered every aspect of the design and achieved all design goals as shown is Table 6.1. The availability of clear user commands and a common API will not only allow users to run SAGEFed supported commands but will also execute client-based commands. So, for example, to list an instance on Savi, the user can enter *./SAGEFed.sh listinstance savi location* or they can use the client tool command *nova list* with appropriate location in the config file. System development for further extensions is very straightforward, as this tool is built using a shell script that can be easily modified in future to accommodate any changes. After the successful launch of SAGEFed at three conferences we believe it holds great promise to be used it will be used by all users across SAVI and GENI.

Design Goal	Status
Propose a minimal overhead as desirable as corresponded to directly using the client tool.	✓
Support user flexibility to utilize both client tool commands and SAGEFed commands.	✓
Let other client tools be integrated quickly in future.	✓

Table 6.1: Design goals and their status

Appendix A

Additional Information

A.1 Prerequisites

Please follow these steps as a part of Prerequisites if you want to use SAGEFed.

- Make sure you have SSH keys and SSL certificates in the GENI Portal. You can verify this by browsing to the Profile tab on the Portal and looking under SSH Keys and SSL.
- **Getting a SAVI account:** On the GENI Portal, scroll down on the Home page until you see Tools. Press the SAVI button.
- Proceed through the remaining prompts until you get an email with your SAVI username and password.
- Finally, go to http://portal.savitestbed.ca/ and login to verify that your credentials work.
- **Getting an Omni bundle:**In the GENI portal, browse to the Profile tab and then select Configure Omni. If necessary, follow the prompts to generate an SSL certificate. Then, click the Download your omni data button.

A.2 Tutorial

1. Install and configure the omni and GENI-SAVI federation tools

    - Use scp to transfer the omni.bundle file you downloaded in the pre-work from your local machine to the Downloads folder on client1.savitestbed.ca.
        - From the folder containing omni.bundle, run: scp omni.bundle savi-username@client1.savitestbed.ca:Downloads
        - When prompted, enter your SAVI username and password.
        - Tip: Windows users should use an SCP client of their choice (such as winscp)
    - Using your SAVI credentials, log in to client1.savitestbed.ca using ssh.
        - Tip: Any SSH tool can be used for this, including the built-in terminal tools on any Unix- or Linux-based system, or the Putty and cygwin ssh on Windows.
        - On UNIX-like systems do: ssh savi-username@client1.savitestbed.ca
        - When prompted, enter your SAVI password.
    - Once you are logged in, configure the omni tool.
        - omni-configure
    - omni-configure should place your SSH keys (geni_key_portal and geni_key_portal.pub in your ∼ /.ssh folder on client1.savitestbed.ca.
        - Check to make sure that the keys are in the correct location:
        - ls ∼/.ssh
        - geni_cert_portal_key geni_cert_portal_key.pub geni_key_portal geni_key_portal.pub
        - id_rsa.pub known_hosts

- Now download and unpack the GENI-SAVI Federation Tool
  - wget http://web.uvic.ca/∼sushilb/federation/tutorial.tar
  - tar xvf tutorial.tar

2. Create a slice on GENI and reserve some virtual machines in it

- Now create a slice on GENI. Use gs-yourinitials as the slice name.
  - Change into the tutorial directory:
  - cd tutorial
  - In the tutorial directory, run:
  - ./tutorial.sh createslice geni slice_name
  - When prompted, enter your SAVI username, password, and tenant.
- Now add a VM running Ubuntu 14 at the InstaGENI Rack assigned to you.
  - The general form of the command to create a VM on GENI is:
  - ./tutorial.sh createvm geni slice-name rack nickname os_image_name
  - To create the virtual machine for the exercise, run:
  - ./tutorial.sh createvm geni slice-name rack nickname Ubuntu-14-04
  - This will take about a minute. It will then come back with a response of the form:– Result Summary: Got Reserved resources RSpec from ...
  - To connect to the created VM please use the hostname="name.rack domain name"
  - The machine will now be in a booting state. It will take about 5-10 minutes before you can log in. We will use the time productively and create a SAVI VM while we wait.

3. Create a virtual machine on SAVI

   - Now add a VM running Ubuntu 14 at the SAVI site assigned to you.
     - The general form of the command to create a VM on SAVI is
     - ./tutorial.sh createvm savi tenant_name location os_image_name vm-size ssh_key vm-name
     - Each attendee will be given a specific site and values for all parameters. The vm-name should be your GENI username, followed by the site-name. e.g., for rickmcg, the name at Toronto will be rickmcg-toronto
     - Use the following command to reserve your SAVI node:
     - ./tutorial.sh createvm savi geniUsers savi-site Ubuntu-14-04-64 small geni_key_portal geni_username-savi-site
   - You should now have one VM on a GENI rack and one VM at a SAVI site.
     - Check the status of your resources on both GENI and SAVI:
     - ./tutorial.sh listinstance geni slice-name
     - ./tutorial.sh listinstance savi savi-site
   - Record the hostname for your GENI VM and the public IP address listed for your SAVI VM as these will be used in the next step. Notice that access to the SAVI machines are by IP address.

4. Set up Ansible for your experiment

   (a) Create an Ansible inventory file

   - An Ansible inventory file is of the form
   - group_name server1_spec server2_spec

- where group_name is a name for a group of nodes, and a server spec contains login information for a node. An example of a server specification is:
- ansible_ssh_host=142.150.208.146 ansible_ssh_port=22 ansible_ssh_user =rickmcg_geni_ansible_ssh_private_key=∼/.ssh/geni_key_portal
- Create an Ansible inventory file named ansible-hosts for your slice.
  - Your ansible-hosts file should look like this:
  - nodes geni_resource_name ansible_ssh_port=22 ansible_ssh_user= our_geni_usernameansible_ssh_key =∼/.ssh/geni_key _portalsavi_resource_ip ansible_ssh_port=22 ansible_ssh_user=ubuntu ansible_ssh_key= ∼/.ssh/geni_key_portal
  - Where geni_resource_name and savi_resource _ip are what you found from the listinstance commands from step 3.

(b) Add private key to SSH agent

- In the next step, Ansible will try to SSH into multiple nodes.
- To avoid entering the passphrase for your private key multiple times, add your private key to your ssh agent as follows:
- ssh-agent bash
- ssh-add ∼/.ssh/geni_key_portal

(c) The ping module

- The ping module simply tries to do a SSH login to a node and reports success or failure.
- Run the following command on your controller:
- ansible nodes -i ansible-hosts -m ping

- If you dont see success everywhere then there is something wrong with your setup. Ask one of the tutorial leaders for help.

(d) The shell module

- The shell module lets you run arbitrary SSH commands in parallel across a set of hosts. It is useful for poking around, or if there is no Ansible module with the functionality you need.
- Try it out:
- ansible nodes -i ansible-hosts -m shell -a "hostname"
- You can replace hostname above with any other Linux command.

(e) The setup module

- The setup module gathers a bunch of information about each node and saves it in variables that you can reference in your Ansible playbook. This will be really useful for this tutorial!
- Try it out on a node to see what it collects (replace your-vm with your hostname):
- ansible your-vm -i ansible-hosts -m setup

5. Create and run an Ansible playbook to install the software you will need

- Now you will use a playbook to install the software you will need on all the nodes.
- hosts: nodes remote_user: root sudo: yes tasks: - name: Update apt cache apt: update_cache=yes

  - name: Install dnsutils (for dig) apt: name=dnsutils

  - name: Install geoip-bin (for geoiplookup) apt: name=geoip-bin

  - name: Install curl apt: name=curl

- To download this playbook to your client machine, type:
- wget http://groups.geni.net/geni/ raw-attachment/wiki/GENIExperimenter /Tutorials/GENI-SAVI/DesignSetup/ software-install-solution.yaml
- Run this playbook on your Ansible control machine against all the nodes in your slice.
- ansible-playbook -i ansible-hosts software-install-solution.yaml

Bibliography

[1] GENI Engineering Conference 23. Geni-savi federation. `http://groups.geni.net/geni/wiki/GEC23Agenda/GENISAVI`, June 2015.

[2] Faisal Alshuwaier, Abdullah A Alshwaier, Ali M Areshey, et al. Applications of cloud computing in education. In *Computing and Networking Technology (ICCNT), 2012 8th International Conference on*, pages 26–33. IEEE, 2012.

[3] Samuel V Angiuoli, Malcolm Matalka, Aaron Gussman, Kevin Galens, Mahesh Vangala, David R Riley, Cesar Arze, James R White, Owen White, and W Florian Fricke. Clovr: a virtual machine for automated and portable sequence analysis from the desktop using cloud computing. *BMC bioinformatics*, 12(1):356, 2011.

[4] Michael Armbrust, A Fox, R Griffith, AD Joseph, RH Katz, A Konwinski, G Lee, DA Patterson, A Rabkin, I Stoica, et al. February 2009,above the clouds: A berkeley view of cloud computing,. Technical report, Technical Report Number UCB/EECS-2009-28, University of California at Berkeley, Electrical Engineering and Computer Science.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica,

et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[6] Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, Monte Goode, and Kate Keahey. Identity federation and attribute-based authorization through the globus toolkit, shibboleth, gridshib, and myproxy. In *5th Annual PKI R&D Workshop*, volume 4, 2006.

[7] Andy Bavier, Yvonne Coady, Tony Mack, Chris Matthews, Joe Mambretti, Rick McGeer, Paul Mueller, Alex Snoeren, and Marco Yuen. Genicloud and transcloud. In *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit*, pages 13–18. ACM, 2012.

[8] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In vini veritas: realistic and controlled network experimentation. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 3–14. ACM, 2006.

[9] Andy C Bavier, Mic Bowman, Brent N Chun, David E Culler, Scott Karlin, Steve Muir, Larry L Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating systems support for planetary-scale network services. In *NSDI*, volume 4, pages 19–19, 2004.

[10] Boualem Benatallah and Hamid R Motahari Nezhad. Service oriented architecture: Overview and directions. In *Advances in Software Engineering*, pages 116–130. Springer, 2008.

[11] Fran Berman, Geoffrey Fox, and Anthony JG Hey. *Grid computing: making the global infrastructure a reality*, volume 2. John Wiley and sons, 2003.

[12] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[13] Rajkumar Buyya. High performance cluster computing. *New Jersey: F'rentice*, 1999.

[14] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. How to enhance cloud architectures to enable cross-federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 337–345. IEEE, 2010.

[15] Computingcloud. Url:. `https://computinginthecloud.wordpress.com/2008/09/25/utility-cloud-computingflashback-to-1961-prof-john-mccarthy/`.

[16] Mache Creeger. Cloud computing: An overview. *ACM Queue*, 7(5):2, 2009.

[17] Juan Angel Lorenzo del Castillo, Kate Mallichan, and Yahya Al-Hazmi. Openstack federation in experimentation multi-cloud testbeds. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 51–56. IEEE, 2013.

[18] Marios D Dikaiakos, Dimitrios Katsaros, Pankaj Mehra, George Pallis, and Athena Vakali. Cloud computing: Distributed internet computing for it and scientific research. *Internet Computing, IEEE*, 13(5):10–13, 2009.

[19] Chip Elliott. Geni-global environment for network innovations. In *LCN*, page 8, 2008.

[20] Yi Fang, I-Lung Kao, Ivan Matthew Milman, and George Conerly Wilson. Single sign-on (sso) mechanism personal key manager, June 5 2001. US Patent 6,243,816.

[21] Serge Fdida, Timur Friedman, and Thierry Parmentelat. Onelab: An open federated facility for experimentally driven future internet research. In *New Network Architectures*, pages 141–152. Springer, 2010.

[22] SAVI-GENI Federation. Experiment. `http://www.savinetwork.ca/blog/2015/06/21/savi-2015-annual-general-meeting-program-is-now-available/`, July 2015.

[23] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *Network and parallel computing*, pages 2–13. Springer, 2005.

[24] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[25] Vincent A Fusaro, Prasad Patil, Erik Gafni, Dennis P Wall, and Peter J Tonellato. Biomedical cloud computing with amazon web services. *PLoS Comput Biol*, 7(8):e1002147, 2011.

[26] GENI. Global environment for network innovations. `http://groups.geni.net/geni`.

[27] Abhishek Gupta and Dejan Milojicic. Evaluation of hpc applications on cloud. In *Open Cirrus Summit (OCS), 2011 Sixth*, pages 22–26. IEEE, 2011.

[28] Rackspace Hosting and NASA. Openstack documentation. `http://docs.openstack.org/`, May 2010.

[29] John Hughes and Eve Maler. Security assertion markup language (saml) v2. 0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, pages 29–38, 2005.

[30] Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia. Savi testbed: Control and management of converged virtual ict resources. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 664–667. IEEE, 2013.

[31] Joon-Myung Kang, Thomas Lin, Hadi Bannazadeh, and Alberto Leon-Garcia. Software-defined infrastructure and the savi testbed. In *Testbeds and Research Infrastructure: Development of Networks and Communities*, pages 3–13. Springer, 2014.

[32] Alan H Karp. Authorization-based access control for the services oriented architecture. In *Creating, Connecting and Collaborating through Computing, 2006. C5'06. The Fourth International Conference on*, pages 160–167. IEEE, 2006.

[33] Alan H Karp, Harry Haury, and Michael H Davis. From abac to zbac: the evolution of access control models. In *Proceedings of the 5th International Conference on Information Warfare and Security, ed. EL Armistead*, pages 202–211, 2010.

[34] Stratos Keranidis, Dimitris Giatsios, Thanasis Korakis, Iordanis Koutsopoulos, Leandros Tassiulas, Thierry Rakotoarivelo, and Thierry Parmentelat. Experimentation in heterogeneous european testbeds through the onelab facility: The case of planetlab federation with the wireless nitos testbed. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 338–354. Springer, 2012.

[35] Tobias Kurze, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai, and Marcel Kunze. Cloud federation. In *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*, 2011.

[36] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009.

[37] Jun Li and Alan H Karp. Access control for the services oriented architecture. In *Proceedings of the 2007 ACM workshop on Secure web services*, pages 9–17. ACM, 2007.

[38] Jun Li and Alan H Karp. Zebra copy: A reference implementation of federated access management. In *ACM Workshop on Secure Web Services, Fairfax, VA November*, 2007.

[39] Rick Mcgeer and Matt Hemmings. The ignite distributed collaborative visualization system. `https://sites.google.com/site/sigmetricsdcc2015/`, June 2015.

[40] RL Morgan, Scott Cantor, Steven Carmody, Walter Hoehn, and Ken Klingenstein. Federated security: The shibboleth approach. *Educause Quarterly*, 27(4):12–17, 2004.

[41] AB MySQL. Mysql, 2001.

[42] Akihiro Nakao, Ryota Ozaki, and Yuji Nishida. Corelab: an emerging network testbed employing hosted virtual machine monitor. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 73. ACM, 2008.

[43] Nova. Url:. `http://nova.OpenStack.org/runnova/index.html`.

[44] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-

computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

[45] Omni. Url:. `http://trac.gpolab.bbn.com/gcf/wiki/Omni`.

[46] OneLab. Description about onelab. `http://www.onelab.eu`.

[47] Openstack. Basic architecture of openstack. `http://usercontent1.hubimg.com/`.

[48] Xinming Ou, Anna Squicciarini, Sebastien Goasguen, and Elisa Bertino. Authorization strategies for virtualized environments in grid computing systems. 2006.

[49] Parth H Pathak and Rudra Dutta. A survey of network design problems and joint design approaches in wireless mesh networks. *Communications Surveys & Tutorials, IEEE*, 13(3):396–428, 2011.

[50] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *ACM SIGCOMM Computer Communication Review*, 33(1):59–64, 2003.

[51] Larry Peterson, Robert Ricci, Aaron Falk, and Je_ Chase. Slice-based federation architecture. *Ad Hoc Design Document (July 2008)*, 1, 2010.

[52] Larry Peterson and Timothy Roscoe. The design principles of planetlab. *ACM SIGOPS operating systems review*, 40(1):11–16, 2006.

[53] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. Omf: a control and management framework for networking testbeds. *ACM SIGOPS Operating Systems Review*, 43(4):54–59, 2010.

[54] Rajiv Ranjan, Aaron Harwood, Rajkumar Buyya, et al. Grid federation: An economy based, scalable distributed resource management system for large-scale resource coupling. *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia*, 2004.

[55] David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16. ACM, 2006.

[56] Naidila Sadashiv and SM Dilip Kumar. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science & Education (ICCSE), 2011 6th International Conference on*, pages 477–482. IEEE, 2011.

[57] Vijay Sarathy, Purnendu Narayan, and Rao Mikkilineni. Next generation cloud computing architecture: Enabling real-time dynamism for shared distributed physical infrastructure. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 48–53. IEEE, 2010.

[58] Eric E Schadt, Michael D Linderman, Jon Sorenson, Lawrence Lee, and Garry P Nolan. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9):647–657, 2010.

[59] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, 2012.

[60] Nabil Sultan. Cloud computing for education: A new dawn? *International Journal of Information Management*, 30(2):109–116, 2010.

[61] Anna Thalassinos. Python learning environment.

[62] Trident. Savi-geni federation. `http://tridentcom.org/2015/show/tutorials`, July 2015.

[63] Wil MP van der Aalst and Mathias Weske. The p2p approach to interorganizational workflows. In *Advanced Information Systems Engineering*, pages 140–156. Springer, 2001.

[64] Von Welch, Tom Barton, Kate Keahey, and Frank Siebenlist. Siebenlist attributes, anonymity, and access: Shibboleth and globus. integration to facilitate grid collaboration. In *In 4th Annual PKI R&D Workshop (To appear*. Citeseer, 2005.

[65] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, 2002.

[66] David L Wood, Derk Norton, Paul Weschler, Chris Ferris, and Yvonne Wilson. Single sign-on framework with trust-level mapping to authentication requirements, May 10 2005. US Patent 6,892,307.