# Design of a Dynamic Provisioning System for a Federated Cloud and Bare-metal Environment

Gregor von Laszewski*, Hyungro Lee, Javier Diaz, Fugang Wang,
Koji Tanaka, Shubhada Karavinkoppa, Geoffrey C. Fox, Tom Furlani

Indiana University
2719 East 10th Street
Bloomington, IN 47408. U.S.A.
*+ 812 822 1311
* laszewski@gmail.com

## ABSTRACT

We present the design of a dynamic provisioning system that is able to manage the resources of a federated cloud environment by focusing on their utilization. With our framework, it is not only possible to allocate resources at a particular time to a specific Infrastructure as a Service framework, but also to utilize them as part of a typical HPC environment controlled by batch queuing systems. Through this interplay between virtualized and non-virtualized resources, we provide a flexible resource management framework that can be adapted based on users' demands. The need for such a framework is motivated by real user data gathered during our operation of FutureGrid (FG). We observed that the usage of the different infrastructures vary over time changing from being over-utilized to underutilize and vice versa. Therefore, the proposed framework will be beneficial for users of environments such a FutureGrid where several infrastructures are supported with limited physical resources.

## Categories and Subject Descriptors

D.4.8 [Performance]: Operational Analysis, Monitors, Measurements D.4.7 [Organization and Design]: Distributed systems

## General Terms

Management, Measurement, Performance, Design, Economics,

## Keywords

Cloud Metric, Dynamic Provisioning, RAIN, FutureGrid, Federated Clouds, Cloud busting, Cloud shifting.

## 1. INTRODUCTION

Batch, Cloud and Grid computing build the pillars of todays modern scientific compute environments. Batch computing has traditionally supported high performance computing centers to better utilize their compute resources with the goal to satisfy the many concurrent users with sophisticated batch policies utilizing a number of well managed compute resources. Grid Computing and its predecessor metacomputing elevated this goal by not only introducing the utilization of multiple queues accessible to the users, but by establishing virtual organizations that share resources among the organizational users. This includes storage and compute resources and exposes the functionality that users need as services. Recently, it has been identified that these models are too restrictive, as many researchers and groups tend to develop and deploy their own software stacks on computational resources to build the specific environment required for their experiments. Cloud computing provides here a good solution as it introduces a level of abstraction that lets the advanced scientific community assemble their own images with their own software stacks and deploy them on large numbers of computational resources in clouds. Since a number of Infrastructure as a Service (IaaS) exist, our experience [1] tells us the importance of offering a variety of them to satisfy the various user community demands. In addition, it is important to support researchers that develop such frameworks further and may need more access to the compute and storage hardware resources than is provided by the current IaaS frameworks. For this reason, it is also important to provide users with the capabilities of staging their own software stack. Recently a number of test-beds have been created that allow the provisioning of software stacks by users. This includes OpenCirrus [2], EmuLab [3], Grid5000 [4] and FutureGrid [5]. Within FutureGrid we developed a sophisticated set of services that simplify the instantiation of images that can be deployed on virtualized and non-virtualized resources.

The work described here significantly enhances the services developed and described in our publications about FutureGrid focusing on dynamic provisioning supported by image management, generation, and deployment [1].

In this paper, we enhance our services in the following aspects

a) Implementation of a uniform cloud metric framework for Eucalyptus 3 and OpenStack Essex.

b) Design of a flexible framework that allows resource re-allocation between various IaaS frameworks, as well as bare metal.

c) Design a meta-scheduler that re-allocates resources based on metric data gathered from the usage of different frameworks.

d) Targeted prototype development and deployment for FutureGrid.

The paper is organized as follows. In Section 2, we introduce the current state of Cloud Metrics as used in various IaaS frameworks. We also introduce the efforts that

## 2. Accounting Systems

Accounting systems have been put into production since the early days of computing stemming back to the mainframe, which introduced batch processing, but also virtualization. The purpose of such an accounting system is manifold, but one of its main purposes is to define a policy that allows the shared usage of the resources:

- *Enable tracking of resource usage* so that an accurate picture of current and past utilization of the resources can be determined and become an input to determining a proper resource policy.
- *Enable tracking of jobs and service usage by user and group* as they typically build the common unit of measurement in addition to the wall clock time as part of a resource allocation policy.
- *Enable a metric for economic charge* so that it can be integrated into resource policies as one input for scheduling jobs within the system.
- *Enable a resource allocation policy* so that multiple users can use the shared resource. The policy allows users to get typically a quota and establishes a priority order in which users can utilize the shared resource. Typically a number of metrics are placed into a model that determines the priority and order in which users and their jobs utilize the resource.
- *Enable the automation of the resource scheduling* task to a systems service instead of being conducted by the administrator.

One of the essential ingredients for such an accounting system are the measurements and metrics that are used as input to the scheduling model and is part of the active computer science research since 1960 with the advent of the first mainframes.

### 2.1 High Performance Computing

As High Performance Computing (HPC) systems have always been shared resources, batch systems usually include an accounting system. Typically metrics that are part of scheduling policies include number of jobs run by a user/group at a time, overall time used by a user/group on the HPC system, wait time for jobs to get started, size of the jobs, scale of the jobs, and more. Many batch systems are today available and include popular choices such as Moab which originated from Maui, SLURM [6], Univa Grid Engine [7] which originated from CODINE [8], PBS [9], LSF [9]. Recently many of these vendors have made access to manipulation of the scheduling policies and the resource inventory, managed by the schedules, much easier by adding Graphical user interfaces to them [9-11]. Many of them have also added services that provide cloud-bursting capabilities by submitting jobs for example to private or public clouds such as AWS.

One of the more popular accounting systems with the community is Gold [?]. Gold introduces an economical charge model similar to that of a bank. Transactions such as deposits, charges, transfers, and refunds allow easy integration with scheduling tools. One of the strength of Gold was its free availability and the possibility to integrate it with Grid resources. Unfortunately, the current maintainers of Gold have decided to discontinue its development and instead provide an alternative as a paid solution. It has to be seen if the community will continue picking up Gold or if they switch to the new system.

One of the most interesting projects that has recently been initiated is the XDMod project [12] that is funded by NSF XD and is integrated into the XSEDE project. One of the tasks of this project includes the development of a sophisticated framework for analyzing account and usage data within XSEDE. However, we assume this project will develop an open source version that can be adapted for other purposes. This component contains an unprecedented richness of features to view, and creates reports based on user roles and access rights. It also allows the export of the data through Web services.

### 2.2 Grids

Accounting systems in Grids were initially pretty separated and not meant to be based on a centralized accounting system. Instead, each member of a virtual organization is pretty separate and has the right to assign and implement their own allocation and accounting policies. This can be backed up by the creation of the earliest virtual organization termed GUSTO [13], but also in more recent efforts such as the TeraGrid [14, 15], XSEDE [16]endnote, and the OpenScience Grid [17]. Efforts were put in place later to establish a common resource usage unit to allow trading between resources, as for example in TeraGrid and XSEDE. The earliest metric to establish usage of Grid services outside of such frameworks in an independent fashion was initiated by [18] for the usage of GridFTP and later on enhanced and integrated by the Globus project for other Grid services such as job utilization. Other systems such as Nimrod [19] provided a platform to the users in the Grid community that introduced economical metrics similar to Gold and allowed for the creation of trading and auction based systems. They have been followed up by a number of research activities [20] but such systems have not been part of larger deployments in the US.

### 2.3 Clouds

The de facto standard for clouds has been introduced by Amazon Web Services [21]. Since the initial offering, additional IaaS frameworks have become available to enable the creation of privately managed clouds. As part of these offering, we have additional components that address accounting and usage metrics. We find particularly relevant the work conducted by Amazon [22], Eucalyptus [23], Nimbus [24], OpenStack [25], and OpenNebula [26]. Other ongoing community activities also contribute in the accounting and metric area, most notably by integrating GreenIT [27] [28].

In addition to the build in monitoring and accounting features, some cloud platforms can be enhanced by existing *monitoring* tools that are well known within the HPC community. Such tools include Nagios [29] and Ganglia [30], which are both open source.

For IaaS frameworks we make the following observations.

*Amazon CloudWatch* [22] provides real-time monitoring of resource utilization such as CPU, disk and network. It also enables users to collect metrics about AWS resources, as well as publish custom metrics directly to Amazon CloudWatch. Amazon CloudWatch functionality is accessible via API, command-line tools, the AWS SDK, and the AWS Management Console. It provides a facility to create alarms on any of users metrics to receive notifications or take other automated actions when the metric crosses user specified threshold. given threshold over a number of time periods. It provides statistics based on metric data. Statistics for a metric can be obtained in

different ways like Statistics Aggregated Across All Instances, Statistics Aggregated by Auto Scaling Group, Statistics Aggregated by Image (AMI) ID, and Statistics for a Specific EC2 Instance.

Users can view graphs and statistics for any of their metrics, and get a quick overview of their alarms and monitored AWS resources in one location on the Amazon CloudWatch dashboard. Amazon CloudWatch integrates with AWS Identity and Access Management (AWS IAM) to control user access to AWS account. It makes easier to specify which CloudWatch actions a user or group in AWS account can perform. Permissions granted using IAM cover all the cloud resources used with CloudWatch.

*Eucalyptus* enables since version 3.0, usage reporting as part of its resource management [31]. However a sophisticated accounting systems are not yet provided that lets users and administrators observe details about particular VM insatnces. Eucalyptus supports monitoring features with popular third party applications such as Nagios and Ganglia to help measuring VM instances and storage utilization since version 1.6 [23]. Nagios typically allows checking the health of cloud clusters and Ganglia provides resource utilization such as CPU, memory, and load average. Eucalyptus also provides audit trails for user requests and resource usage via log messages. The log files can be parsed and analyzed for generating usage statistics in terms of the cloud metric and for more information, in the debug mode of logs, traceable events of VM instances, storages and networks are recorded for performing proper measurement. However, a convenient framework that provides such features to the users is not provided by default. We have developed as part of our effort described here such a framework that is made available as open source project [32]. In addition to Eucalyptus we can also depict usage data from OpenStack, thus our framework targets federated cloud environments with heterogeneous cloud infrastructures rather than a single IaaS framework.

*Nimbus* claims per-client usage tracking and per-user storage quota in cumulus (the VM image repository manager for Nimbus) as accounting features. The per-client usage tracking provides information of requested VM instances and historical usage data. The per-user storage quota enables restriction of file system usage. Nimbus also uses Torque resource manager for gathering accounting logs. For monitoring features, Nimbus utilizes Nagios and Cloud Aggregator, which is a utility to receive system resource information.

*OpenNebula* has a utility named OpenNebula Watch [26] as an accounting information module. It stores activities of VM instances and hosts (clusters) to show resource utilization or charging data based on the aggregated data. OpenNebula Watch requires database handler like sequel, sqlite3 or MySQL to store the accounting information. It checks the status of hosts so physical systems can be monitored, for example, CPU and memory except network.

*OpenStack* is currently under heavy development in regards to many of its more advanced components. An on-going effort for developing accounting systems of OpenStack exists which is named Efficient Metering or ceilometer. It aims to collect all events from OpenStack components for billing and monitoring purposes [33]. This service will measure general resource attributes such as CPU core, memory, disk and network as used by the nova components. Additional metrics might be added to provide customization. Efficient Metering is planned to be released in the next version of Openstack (Folsom) late in 2012. Besides this effort, other metric projects esist and include several billing projects such as [34], [35], and [35].

*Microsoft Azure* The System Center Monitoring Pack for Windows Azure applications is the most cost effective and flexible platform for managing traditional data centers, private and public clouds, and client computers and devices [36]. It is the only unified management platform where multiple hypervisors, physical resources, and applications can be managed in a single offering. From a single console view, the IT assets like network, storage and compute can be organized into a hybrid cloud model spanning the private cloud and public cloud services. By default, the monitoring is not enabled. Therefore, the discovery must be configured by using the Windows Azure Application monitoring template for each Windows Azure Application to be monitored. The performance monitoring can also be enabled by using some tools like Powershell cmdlets for Windows Azure [37] and Azure Diagnostics Manager 2 from Cerebrata [38]. The monitoring data can be visualized using System Center Operation Manager Console. The monitoring pack provides functionalities such as discover Windows Azure applications, providing status for each role instance, collecting and monitoring performance information, collecting and monitoring windows events, collecting an d monitoring the .NET framework trace messages from each role instance, grooming performance, events and the .NET framework trace data from Windows Azure storage account and changing the number of role instances. The monitoring pack provides Operation Manafer GUI to view all the metrics and graphs. From Operation Manager, user can create custom dashboard or publish graphs on SharePoint to people who do not have the SCOM console.

*Google Compute Engine* is a IaaS product launched end of June, 2012 apart from Google App Engine which is a Google's PaaS cloud platform [39]. Some features are limited due to ongoing development. Google currently supports several options for networking and storage while managing virtual machines through the compute engine. Presently, there is no accounting APIs for Google Compute Engine, but there a monitoring API for Google App Engine exists. Google App Engine supports a usage report for displaying resource utilization of instances in the administration console [40] and provides a runtime API [41] to retrieve measured data from the application instances such as CPU, memory, and status. We expect that similar functionality will become available for the Google Compute Engine as well.

## 3. FutureGrid A TestBed suitable for Federated Cloud Research

FutureGrid [42] provides a set of distributed resources totaling about 3000 compute cores. Resources include a variety of different platforms allowing users to access heterogeneous distributed computing, network, and storage resources. Services to conduct HPC, Grid, and Cloud projects including various IaaS and PaaS are offered. This variety of resources and services allow interesting interoperability and scalability experiments that foster research in the area of for example federated clouds. Users can experiment with various IaaS frameworks at the same time, but also integrate Grid and HPC services that are of especial interest to the scientific community. One important feature of the FutureGrid software services is the ability to dynamically provision resources not only by using virtualization technologies, but also by dynamically provision on bare-metal [1]. This feature allows us to *rain* not only a software stack onto an OS that is hosted on a resource, but also to replace the entire

OS onto the compute server. Authorized users have access to this feature that is ideal for performance experiments. Via the help of Rain we can now *shift* compute servers into various clouds determined by user demand.

# 4. REQUIREMENTS

Within the [1] we presented qualitative and quantitative evidence that users are experimenting with a variety of IaaS frameworks. To support this need, we have instantiated multiple clouds on distributed compute clusters in FG. Furthermore, we are able to support multiple IaaS frameworks on our resources. However, the association of compute servers to the various IaaS frameworks is conducted currently by hand through a best effort attempt by the system administrators. From our experience cast from a variety of projects, we have seen some interesting use patterns. One such use pattern arises from educational classes in the distributed computing area. In this case we observe that classes cycle through topics to teach students about HPC, Grid, and Cloud computing. When teaching cloud computing they also introduce multiple cloud frameworks. Thus, the demand to access the resources one after another is a logical consequence based on the way such classes are taught.

On the other hand, we observe some projects that utilize the resources in a federated fashion either while focusing on federation within the same IaaS framework [43], but more interestingly also to federate between IaaS frameworks while focusing on scientific workflows that utilize cycle scavenging [44] or select frameworks that are most suitable for a particular set of calculations as part of the workflow [45].

As a consequence of our observations, we derived the following requirements that will shape the design of the services that support cloud federation.

- *Support for multiple IaaS*: This includes OpenStack, Nimbus, Eucalyptus, and OpenNebula. Furthermore we like to integrate with AWS and potentially other clouds hosted outside of FG.
- *Support for bare-metal provisioning to the privately managed resources*: This will allow us the to *rain* custom designed software stacks on OS we choose onto each of the servers on demand.
- *Support for dynamic adjustment of service assignments*: The services placed on a server are not fixed, but can change over time via rain.
- *Support for educational class patterns*: Compute classes often require a particular set of services that are accessed by many of its members concurrently leading to spikes in the demand for one service type.
- *Support for advance provisioning:* Sometimes users know in advance when they need a particular service motivating the need for the instantiation of services in advance. This is different from advance reservation of a service, as the service is still shared by the users.
- *Support for advance reservation:* Some experiments require the exclusive access to the services.
- *Support for automation:* Managing such an environment should be automatized as much as possible.
- *Support for inter-cloud federation experiments:* Ability to access multiple IaaS instances at the same time.
- *Support for divers user communities*: Users, Administrators, Groups, and services are interested in using the framework.

We intend to implement this design gradually and verify it on FG. The resulting software and services will be made available open source so others can utilize them also.

Due to these requirements we must support three very important functions of our framework. These functions include:

*Cloud busting*, which enables the instantiation of new cloud frameworks within FutureGrid.

*Cloud shifting*, which enables moving (or re-allocating) compute resources between the various clouds and HPC.

*Resource Provisioning*, which is a basic functionality to enable cloud busting and shifting as it allows the dynamic provisioning of the OS and software stack on bare-metal.

Together these three functions build an enhancement to our rain tool that is important for the work described in this paper.

However, we will not address topics such as cloud bursting, that enables to outsource services in case of over-provisioning, or inter cloud federation enabling to use the compute or storage resources across various clouds, and is sometimes referred to as sky computing.

# 5. DESIGN

Before we explain our architecture we have to point out some features of the resource and service fabric building a central component within our design. We assume that the *Resource Fabric* consists of a resource pool that contains a number of compute services. Such services are provided either as a cluster or as part of a distributed network of workstations (NOW). The resources are grouped based on network connectivity proximity. This will allow the creation of regions within cloud IaaS environments to perform more efficiently among its servers. We assume a rich variety of services offered in the *Service Fabric*. This includes multiple IaaS, PaaS frameworks, and HPC environments. Instead of assuming that there can only be one cloud for a particular IaaS framework, we envision multiple independent clouds could be managed. This assumption allows users potentially to host their own privately managed clouds. Such a model is currently deployed as part of the FutureGrid Operation allowing users to access a variety of preconfigured clouds to conduct interoperability experiments among the same IaaS and also different IaaS frameworks, as well as the inclusion of dedicated HPC services.

Having access to such a comprehensive environment opens up a number of interesting design challenges. We observe that our operational mode is significantly enhanced in contrast to other academic clouds that typically only install a single IaaS framework on their resource [46, 47]. Thus such environments cannot offer by themselves the comprehensive infrastructure needed to conduct many of the topics that arise in cloud federation.

One of the question we need to answer is how we can best utilize such an environment that supports inter-cloud and bare-metal demands posed by the users as we have practically observed in FutureGrid and how we can integrate these requirements into a software architecture.

We designed a software architecture to address the requirements presented earlier. In **Error! Reference source not found.** we present a layered view of this architecture. We distinguish the user layer allowing administrators, but also users (and groups of users) to interact with the framework. In addition we point out that Web services can interact with it to develop 3rd party

automated tools and services leveraging the capabilities. Access to the various functions is provided in secure fashion. Due to the diverse user communities wishing to use the environment, our design supports a variety of access interfaces including command line, dashboard, web services, as well as libraries and APIs.

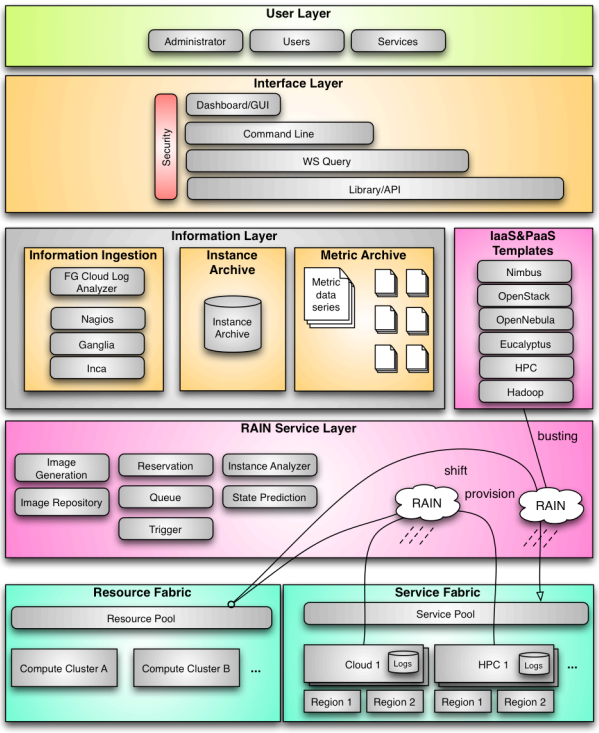


**Figure 1: Design of the rain-based federated cloud management services.**

An important feature is to be able to integrate existing and future information services to provide the data to guide dynamic and automatic resource provisioning, cloud busting, and cloud shifting. Due to this reason we allow in our design the integration of events posted by services such as Inca, Ganglia, and Nagios. Moreover, we obtain information from the running clouds and, when the provided information is not sufficiently, we will be able to ingest our own information by analyzing log files or other information obtained when running a cloud. For clouds we also host an instance archive that allows us to capture traces of data that can be associated with a particular virtual machine instance. A metric archive allows the registration of a self-contained service that analyses the data gathered while providing a data series according to the metric specified. Metrics can be combined and can result in new data series.

At the center of this design is a comprehensive *RAIN* service Layer. Rain is an acronym for *R*untime *A*daptable *IN*sertion service signifying services that on the one hand adapt to runtime conditions and on the other allow to be inserting or dynamically provision software environments and stacks. We use the terms rain and raining to refer to the process of instantiate services on the resource and service fabrics. In this analogy we can *rain* onto a cluster services that correspond to an IaaS, a PaaS, or a HPC batch system. Rain can be applied to virtualized and non-virtualized machine images and software stacks. This is refereed to as Cloud busting.

In addition Rain can also be used to move resources between already instantiated environments, hence supporting cloud-shifting. The most elementary operation to enable cloud busting and cloud shifting is to provision the software and services onto the resources. We have devised this elementary operation and introduced in [48]. In our past effort we focused on the problem of image management, while in this work we extend this work to focus on cloud shifting.

*Image Management.* Rain allows us to dynamically provision images on IaaS and HPC resources. As users need quite a bit of sophistication to enable a cross platform independent image management, we have developed some tools that significantly simplify this problem. This is achieved by creating template images that are stored in a common image repository and adapted according to the environment or IaaS framework in which the image is to be deployed. Hence users have the ability to setup experiment environments that provide similar functionality in different IaaS such as OpenStack, Eucalyptus, Nimbus[1], and HPC. Our image management services support the entire lifecycle including image generation, image storage, image reuse, and image deployment. Furthermore, we started to provide extensions for image usage monitoring and quota management.

*Cloud Shifting.* To enable cloud shifting we have introduced a number of low-level tools and services that allow the de-registration of resources form an IaaS or HPC service and move it over to another. This is done obviously the following steps:

1. Identify which resources should be moved as part of the shift.
2. Deregister the resources from the service they are currently registered with and move them into our available resource pool.
3. Pick resources from the available resource pool and rain the needed OS and other services onto that resource (if not already available)
4. Register the resource with the appropriate service and advertise its availability
5. Use as appropriate within the new framework.

*Cloud Busting*. Cloud busting enhances the previous process by two additional steps.

0.1. Decide which IaaS or HPC services to instantiate
0.2. Set up the new cloud or HPC environment

This is not a simple process, but requires a great deal of planning and knowledge about the available infrastructure. Currently we execute this step by hand but intend to further automatize it as much as possible. However, more details about Cloud busting will be discussed in another upcoming paper that is not yet available.

*Queue Service*. As we anticipate that users may have demands that can not immediately be fulfilled to conduct cloud shifting or

[1] Nimbus is only partially supported based on predefined images, as it does at this time not allow adaptation of kernel and other features that are exposed by other IaaS frameworks.

cloud busting, our design includes the introduction of a queuing service that can coordinate multiple such requests,

*Reservation Service.* As we also expect that we will obtain from users definite requests to be fulfilled at predefined times, such as in the case of tutorials, classes, and regularly executed experiments our design also includes the introduction of a reservation service.

*State Prediction and Service.* A state prediction service will provide input to how our cloud universe will be configured. Our instance database and instance analyzer will provide valuable input for the runtime configuration of the resource and service fabrics. An important aspect of our design is that the prediction service can be augmented and enhanced by various scheduling algorithms as well as the utilization of various metrics. This is of special importance as we expect to analyze usage patterns of real Cloud services provided by FutureGrid.

*Metrics.* An elementary input to our prediction service is are metrics provided to our framework. These metrics are feed by elementary information in regards to job and virtual machine traces.

Traditionally in a computing system, the common resource metrics are CPU, memory, storage, network bandwidth, and electricity and those are on the bills as what users pay for.

In case of VMs, we have to expand this information with VM specific information as such as VM state, size, type, os, memory, disk, CPU, kernel, Network IP, owner, and label. In addition, we are concerned with how much time it costs to create the VM, transfer it to a resource, instantiate and dynamically provision it, as well as bringing it in a state that allows access by the user. Furthermore, once the machine is shut down, we need to account for the shutdown time and eventual cleanup or removal of the VM. Naturally we also need to keep track on which user, group or project instantiated the VM and if the image is a replication run in parallel on other resources in the fabric.

When dealing with services that are dependent on performance metrics we also have to deal with periodicity of the events and filter out events not only based potentially on an a yearly, monthly, weekly, daily, hourly, minute or per second basis, but to eliminate events that do not contribute significantly to the trace of a virtual machine image. We have practically devised such a service for Eucalyptus that reduced 4 million lag events to about 10000 trace events for virtual machine images. This allows us to query needed information for our predictive services in milliseconds rather than hours of reanalyzing such log entries over and over again. Hence our design is not only to retrieve standard information such as average, sum, minimum and maximum, as well as count of VM related events, but can input this data efficiently into a time series analysis and predictive service.

## 6. STATUS AND IMPLEMENTATION

As already pointed out we have developed the basic infrastructure to support rain by enabling the image management services. These services are in detail documented in [?]. Recently we have started the development of rain services that address the issue of cloud shifting. We have already developed the ability to add and remove dynamically resources from and to Eucalyptus clouds. At present we are enhancing these activities for OpenStack and HPC. Once implemented we will be easily be able to move resources between OpenStack, Eucalyptus, and HPC services. Based on our requirements, FutureGrid is also funding the Nimbus project to enhance their services so they allow similar features as other IaaS frameworks in regards to image management. In parallel we have significantly contributed towards the analysis of instance data for Eucalyptus and OpenStack clouds. Such data is instrumental for our predictive services. Such a service was not available as part of the existing development efforts. This effort includes the creation of a comprehensive database for instance traces that records important changes conducted as part of the VM instance runtime documented in our design section. A previous analysis effort that analysis log files in a repeated fashion was replaced with an effort that allows the ingestion and extraction of important log files from newly created log events. As a result we were able to reduce over 4 million log entries to a couple of 10000 important events. The reduction in this data lead to a speedup of our analyze capabilities from hours to milliseconds. In addition we have separated the dependency on a web interface for this activity be introducing a command line interface, a web services, as well as a simple graphical interface to this data (see Figure 3).

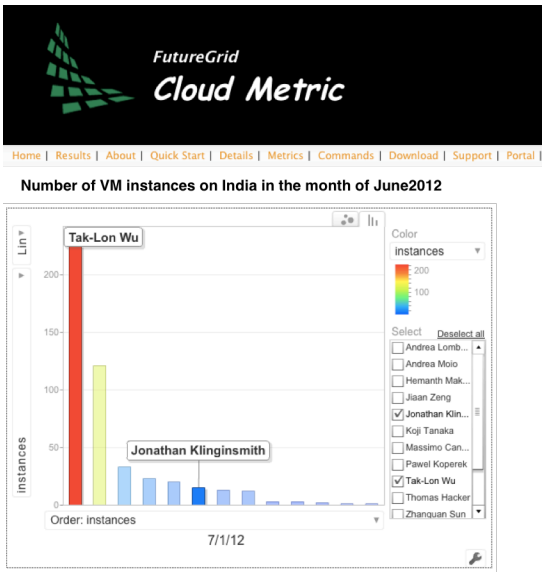


**Figure 3: Screenshot of our Cloud Instance analyzing framework applied to data from FutureGrid for Eucalyptus.**

At the same time the code was significantly reduced and modularized so that future maintenance and enhancements become easier. One of the results is that we even can switch out the library that presents the data to the user through the Web interface. Examples for data currently presented in our We interface is based on the utilization of several metrics. This includes:

- Total running hours of VM instances:
- Total count of VM instances in a particular state:
- CPU cores / Memory / Disk allocations:
- Delay of Launching and termination requests (provisioning Interval [49]
- Geographical locations of VM instances

Additional metrics such as Traffic intensity for a particular time period [50, 51]. Metrics projecting a per user, per group, or per project view, metrics per cloud view, as well as metrics for the overall infrastructure and metrics related to the resource and service fabric are under development.

## 7. CONCLUSION

In this paper we have presented a unique design of a federated cloud environment that is not focused on just supporting a single IaaS framework. In addition we are able to integrate traditional HPC services. This work is based on our services developed previously that have been significantly enhanced by addressing challenges arising in cloud busting and cloud shifting. One of the other contributions of this paper is the creation of a metric framework that allows us to manage traces of virtual machine instances that can be derived from Eucalyptus and OpenStack log files. We are currently continuing our implementation efforts following our design. We welcome additional collaborators to contribute to our efforts and to use FutureGrid.

## 8. Acknowledgement

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 and 1025159. We like to thank the members of FG for their help and support.

## 9. References

[1] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, "Comparison of Multiple Cloud Frameworks," presented at the IEEE CLOUD 2012, 5th International Conference on Cloud Computing, Honolulu, 2012.

[2] I. A. Arutyun, "Open Cirrus: A Global Cloud Computing Testbed," vol. 43, pp. 35-43, 2010.

[3] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," presented at the Proceedings of the 5th Symposium on Operating Systems Design & Implementation, 2002.

[4] (2012). *Grid5000 Home Page*. Available: https://http://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

[5] G. von Laszewski, G. C. Fox, Fugang Wang, A. J. Younge, A. Kulshrestha, G. G. Pike, W. Smith, J. Vöckler, R. J. Figueiredo, J. Fortes, and K. Keahey, "Design of the FutureGrid experiment management framework," presented at the Gateway Computing Environments Workshop (GCE) at SC10, New Orleans, LA, 2010.

[6] A. Yoo, M. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*. vol. 2862, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 44-60.

[7] Univa. *Grid Engine*. Available: http://www.univa.com/products/grid-engine/

[8] Genias. *CODINE: Computing in distributed networked environments (1995)*. Available: http://www.genias.de/genias/english/ codine.html

[9] Altair. *PBS*. Available: http://www.pbsworks.com/

[10] (2012). *Moab*. Available: http://www.adaptivecomputing.com/products/hpc-products/

[11] Bright-Computing. *Cluster Manager*. Available: http://www.brightcomputing.com/Bright-Cluster-Manager.php

[12] XDMoD. *XDMoD (XSEDE Metrics on Demand)*. Available: https://xdmod.ccr.buffalo.edu/

[13] Globus-Project. *Globus Ubiquitous Supercomputing Testbed Organization (GUSTO)*. Available: http://www.startap.net/PUBLICATIONS/news-globus2.html

[14] TerraGrid. *TeraGrid Web Page*. Available: http://www.teragrid.org/

[15] D. Hart, "Measuring TeraGrid: Workload Characterization for an HPC Federation," presented at the Proceedings of IJHPCA, 2011.

[16] (July 15). *XSEDE: Extreme Science and Engineering Discovery Environment*. Available: https://http://www.xsede.org

[17] OSG. *Open Science Grid*. Available: http://www.opensciencegrid.org

[18] G. von Laszewski, J. DiCarlo, and B. Allcock, "A Portal for Visualizing Grid Usage," *Concurrency and Computation: Practice and Experience,* vol. 19, pp. 1683-1692, presented in GCE 2005 at SC'2005 2007.

[19] D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sosic, R. Sutherst, and N. White, "The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing," presented at the Proceedings of the 20th Australasian Computer Science Conference, 1997.

[20] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities, in," 2008.

[21] Amazon. *Amazon Web Services*. Available: http://aws.amazon.com/

[22] Amazon. *Web Services Cloud Watch*. Available: http://docs.amazonwebservices.com/AmazonCloudWatch/latest/DeveloperGuide/CloudWatch_Introduction.html

[23] Eucalyptus. *Eucalyptus Monitoring*. Available: http://open.eucalyptus.com/wiki/EucalyptusMonitoring_v1.6

[24] Nimbus-Project. *Per Client Tracking*. Available: http://www.nimbusproject.org/docs/current/features.html

[25] OpenStack. *Multi-Tenant Accounting*. Available: http://wiki.openstack.org/openstack-accounting?action=AttachFile&do=get&target=accounts.pdf

[26] OpenNebula. *OpenNebula Watch - Accounting and Statistics 3.0*. Available: http://opennebula.org/documentation:archives:rel3.0:acctd_conf

[27] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems," 2010.

[28] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-Efficient Cloud Computing," 2010.

[29] W. Barth, *Nagios: System and Network Monitoring*. San Francisco, CA, USA: No Starch Press, 2006.

[30] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing,* vol. 30, pp. 817 - 840, 2004.

[31] Eucalyptus. *Eucalyptus 3.0.2 Administration guide*. Available: http://www.eucalyptus.com/docs/3.0/ag.pdf

[32] G. v. Laszewski, H. Lee, and F. Wang. *Eucalyptus Metric Framework (Source Code)*. Available: https://github.com/futuregrid/futuregrid-cloud-metrics

[33] OpenStack. *Blue print of Efficient Metering*. Available: http://wiki.openstack.org/EfficientMetering

[34] Z. Luo. *Dough*. Available: https://github.com/lzyeval/dough

[35] OpenStack. *Billing Plugin for OpenStack*. Available: https://github.com/trystack/dash_billing

[36] Microsoft. *Introduction to the Monitoring Pack for Windows Azure Applications*.

[37] Microsoft. *Windows Azure PowerShell Cmdlets*. Available: http://wappowershell.codeplex.com/

[38] Red-Gate-Software. *Cerebrata*. Available: http://www.cerebrata.com/

[39] Google. *Google Compute Engine*. Available: http://en.wikipedia.org/wiki/Google_Compute_Engine

[40] Google. *Monitoring Resource Usage of Google App Engine*. Available: https://developers.google.com/appengine/docs/python/backends/overview#Monitoring_Resource_Usage

[41] Google. *Runtime API for Google App Engine*. Available: https://developers.google.com/appengine/docs/python/backends/runtimeapi

[42] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, G. G. Pike, W. Smith, J. Voeckler, R. J. Figueiredo, J. Fortes, K. Keahey, and E. Deelman, "Design of the FutureGrid experiment management framework," presented at the Gateway Computing Environments Workshop (GCE), 2010 in conjunction with SC10, New Orleans, LA, 2010.

[43] K. Katarzyna. (2009) Sky Computing. 43-51. Available: http://doi.ieeecomputersociety.org/10.1109/MIC.2009.94

[44] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.,* vol. 13, pp. 219-237, 2005.

[45] Thilina Gunarathne, Judy Qiu, and Geoffrey Fox, "Iterative MapReduce for Azure Cloud " presented at the CCA11 Cloud Computing and Its Applications, Chicago, ILL, 2011.

[46] Cornell. *Cornell University Red Cloud*. Available: http://www.cac.cornell.edu/redcloud/

[47] Clemson. *Clemson University One Cloud*. Available: https://sites.google.com/site/cuonecloud/

[48] J. Diaz, G. von Laszewski, F. Wang, and G. Fox, "Abstract Image Management and Universal Image Registration for Cloud and HPC Infrastructures," presented at the IEEE CLOUD 2012, 5th International Conference on Cloud Computing, Honolulu, HI, 2012.

[49] "Report on Cloud Computing to the OSG Steering Committee."

[50] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments," *Parallel Processing, International Conference on,* vol. 0, pp. 295-304, 2011.

[51] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters," 2012.