Design of the FutureGrid Experiment Management Framework

Gregor von Laszewski*, Geoffrey C. Fox*, Fugang Wang*, Andrew J. Younge*, Archit Kulshrestha*,
Gregory G. Pike*, Warren Smith†, Jens Vöckler‡, Renato J. Figueiredo§, Jose Fortes§, Kate Keahey¶
*Pervasive Technology Institute, Indiana University, Bloomington, IN 47408, USA
Email: see https://futuregrid.org
†Texas Advanced Computing Center, University of Texas, 10100 Burnet Road, Austin, TX 78758-4497, USA
‡ISI/USC, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292, USA
§Electrical and Computer Engineering, University of Florida, Gainesville, FL, 32611,USA
¶Computation Institute at the University of Chicago, Chicago IL, USA

***Abstract*—FutureGrid provides novel computing capabilities that enable reproducible experiments while simultaneously supporting dynamic provisioning. This paper describes the FutureGrid experiment management framework to create and execute large scale scientific experiments for researchers around the globe. The experiments executed are performed by the various users of FutureGrid ranging from administrators to software developers and end users. The Experiment management framework will consist of software tools that record user and system actions to generate a reproducible set of tasks and resource configurations. Additionally, the experiment management framework can be used to share not only the experiment setup, but also performance information for the specific instantiation of the experiment. This makes it possible to compare a variety of experiment setups and analyze the impact Grid and Cloud software stacks have.**

I. Introduction

FutureGrid (FG) [1] provides computing capabilities that will enable researchers to tackle complex research challenges related to the use and security of Grids and Clouds. These include topics ranging from authentication, authorization, scheduling, virtualization, middleware design, interface design and cybersecurity, to the optimization of Grid-enabled and cloud-enabled computational schemes for researchers in fields such as astronomy, chemistry, biology, engineering, atmospheric science and epidemiology. FG provides a significant and new experimental computing Grid and Cloud test-bed to the research community, together with user support for third-party researchers conducting experiments on FutureGrid. The test-bed will make it possible for researchers to conduct experiments by submitting an experiment plan, which is then executed via a sophisticated workflow engine, preserving the provenance and state information necessary to reproduce the experiment.

The testbed includes a geographically distributed set of heterogeneous computing systems, a data management system that will hold both metadata and a growing library of software images necessary for Cloud computing, and a dedicated network allowing isolated, secure experiments. The testbed will support virtual machine-based environments, as well as operating systems on native hardware for experiments aimed at minimizing overhead and maximizing performance. The project partners will integrate existing open-source software packages to create an easy-to-use software environment that supports the instantiation, execution and recording of Grid and Cloud computing experiments. Furthermore, with the advent of emerging Cloud technologies, users have a newfound ability to define their own environment specific to their needs using virtualized services.

One of the goals of the project is to understand the behavior and utility of Cloud computing approaches. Recently, Cloud computing has become quite popular and a multitude of Cloud computing middleware solutions now exist. However, it is not clear at this time which of these Cloud middleware tools is preferred amongst both users and administrators. FG provides the ability to compare these frameworks with each other while considering real scientific applications. As such, researchers will be able to measure the overhead of Cloud technology by requesting linked experiments on both virtual and bare-metal systems. This ability provides valuable information that will help users decide which infrastructure suits them best and will also to help users looking to transition from one environment to the other.

Participants. FutureGrid is a large-scale testbed that includes a multitude of participating sites and collaborators. The list of primary participants of FutureGrid includes (see Figure 1) Indiana University, Purdue University, San Diego Supercomputer Center at University of California San Diego, University of Chicago/Argonne National Laboratory, University of Florida, University of Southern California Information Sciences Institute, University of Tennessee Knoxville, Texas Advanced Computing Center at University of Texas at Austin, University of Virginia, Center for Information Services, and GWT-TUD from Technische Universtität Dresden. However, users of FG do not have to be from these partner organizations. Furthermore, we hope that new organizations in academia and industry can partner with the project in the future.

Organization of the Paper. The paper is structured as follows. First we provide more details about the resources that are part of the FG testbed. Then we introduce the concept of dynamic provisioning that motivated our architecture while

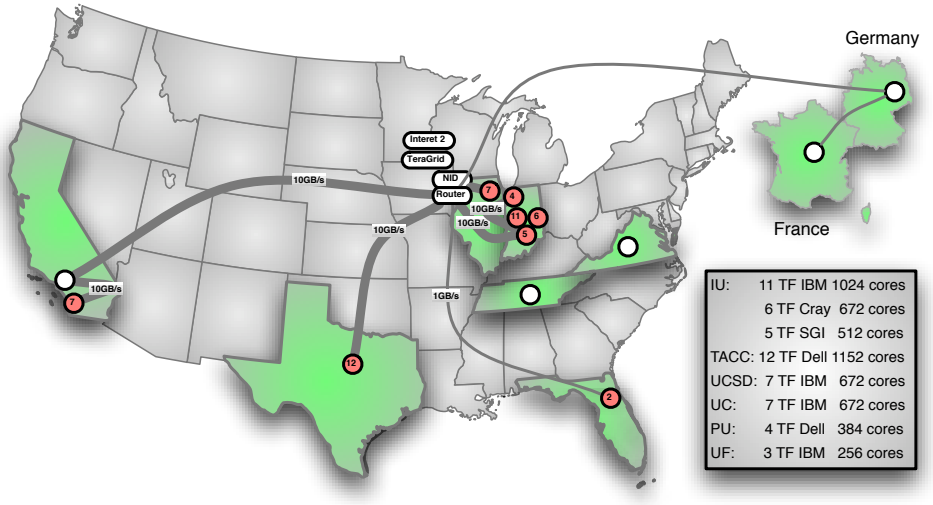

Fig. 1. FutureGrid Participants and Resources

"raining" an environment onto resources. Next we focus on a particular portion of the architecture that is specifically centered on the experiment management framework. We will describe this framework in more detail, while focussing on how we organize experiments, and how images are created and shared among experiment users.

II. FG Facilities

FutureGrid is a national-scale Grid and Cloud test-bed facility that includes a number of computational resources at distributed locations. The FutureGrid network is unique and can lend itself to a multitude of experiments specifically for evaluating middleware technologies and experiment management services. This network can be dedicated to conduct experiments in isolation, using a network impairment device for introducing a variety of predetermined network conditions. Figure 1 depicts the geographically distributed resources that are outlined in Table I in more detail. All network links within FutureGrid are dedicated (10GbE lines for all but to Florida, which is 1GbE), except the link to TACC. That link will initially be shared as part of the TeraGrid network, however we expect to be able to implement dedicated links as needed dynamically once TeraGrid Phase III (XD) is implemented. It is planned that FutureGrid will be connected to an archival storage system that is distributed among a number of sites (see Table II).

Although the total number of systems within FutureGrid is conservative, they provide some heterogeneity to the architecture and are connected by high-bandwidth network links. One important feature to note is that most systems can be dynamically provisioned, e.g. these systems can be reconfigured when needed by special software that is part of FutureGrid with proper access control by users and administrators.

A Spirent H10 XGEM Network Impairment emulator [2], co-located with the core router [3], will provide a central resource to introduce network latency, jitter, loss, and errors to network traffic within FutureGrid. It allows for a delay from 50 ms to 15 seconds with a granularity of 16 ns, and for 0-100% of packet loss and various types of errors with a granularity of 0.0001%. It provides full bidirectional 10G w/64 byte packets, allowing packet manipulation in the first 2000 bytes, and up to 16 k frame sizes. A scripting interface is provided through TCL and a Web interface allows for customization through a portal. Consequently, this device will provide great flexibility in enabling a wide variety of experimental conditions for testing network and Grid applications.

TABLE II
FutureGrid data facilities

System Type	Capacity (TB)	Site
DDN 9550	335	IU
DDN 6620	120	UC
SunFire x4170	72	SDSC
Dell MD3000	30	TACC

III. FG Dynamic Provisioning

The goal of dynamic provisioning is to partition a set of resources in an intelligent way to provide a user defined environment to any user that makes such a request. This entails a specific, specialized deployment which can allocate and deallocate resources in real-time. As such, customized environments need to be in place and be able to dynamically add and remove resources depending on the overall system load and utilization. Dynamic provisioning is used in several contexts as part of FG:

- **Dynamic Resource Assignment.** Resources in a cluster may be reassigned based on the anticipated user require-

TABLE I
FUTUREGRID HARDWARE

System type	# CPUs	# Cores	TFLOPS	RAM (GB)	Storage (TB)	Site
IBM iDataPlex	256	1024	11	3072	†335	IU
Dell PowerEdge	192	1152	12	1152	15	TACC
IBM iDataPlex	168	672	7	2016	120	UC
IBM iDataPlex	168	672	7	2688	72	UCSD
Cray XT5m	168	672	6	1344	†335	IU
Shared mem. system	‡40	‡480	‡4	‡640	‡†335	IU
IBM iDataPlex	64	256	2	768	5	UF
Total	1337	5600	58	10560	552	

†Indicates shared file system. ‡Best current estimate

ments, e.g. a server may be participating as part of an HPC application on the machine, but at a later time the server may be removed from the HPC resource pool and included through dynamic provisioning into a Eucalyptus Cloud. Resources that are not used are in an "unused resource pool".

- **Execution-based Dynamic User Requested Resource Assignment.** At the time of the job execution, a system is provisioned that fulfills the user's needs.
- **Queue-based Dynamic User Requested Resource Assignment.** Since the provisioning of images is time consuming, it is often possible to queue such jobs with the same image requirement in a queue and instantiate the provisioning before all jobs are executed which belong to the queue.

This capability is unique and offers users a new perspective on exploring systems research within a Cloud or Grid computing deployment. In the current implementation of FutureGrid, the dynamic provisioning features are provided by a combination of using XCAT [4] and Moab [5]. As the term dynamic provisioning is not consistently used in the community, we use the term "raining" within the FutureGrid project as a description for placing an environment onto resources. The reason is that our use of dynamic provisioning goes beyond the services offered by common scheduling tools that provide such features. In fact we want our users to rain an HPC, a Cloud environment, or a virtual network onto our resources with little effort. Hence we will provide simple command line tools supporting this task. A recent example within FutureGrid is to "rain" a Hadoop environment defined by a user onto a given cluster. Instead of the user having to learn a complex set of commands that depend on intrinsic functions of the queuing system and low level support software, users can simply invoke a command:

```
fg-hadoop -n 8 -app myHadoopApp.jar ...
```

The dynamic provisioning and scheduling of the job is handled exclusively by the FG system. Users and administrators do not have to set up the Hadoop environment as it is done for them (see Figure 2).

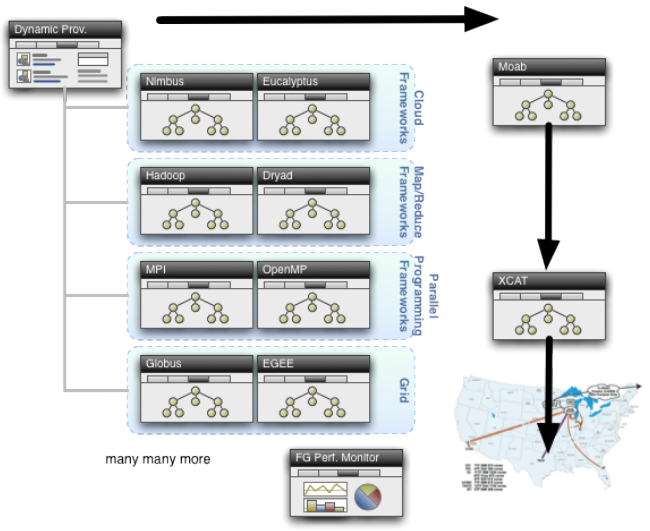

Fig. 2. Dynamic Provisioning in FG allows high level "raining" of systems onto the FG resources.

IV. FG CONCEPTUAL OVERVIEW

In order to achieve this high level of abstraction and to support the rich set of frameworks intended to be introduced by FG, we have developed an extensive architecture as depicted in Figure 3. The architecture is composed of the following components:

FG Hardware. As mentioned earlier, FG is built using a set of NSF sponsored hardware that is geographically distributed among several sites [6]. The hardware includes IBM iDataPlax systems of various sizes, a Dell Cluster, and a Cray XT5m. In addition, we have a dedicated network between the resources which provides the capability of experimenting with network parameters by using a network impairment device.

FG Software. FG is designed to be able to deal with experiments entailing multiple sets of default software stacks

and environments. A user wanting to perform experiments with Unicore, for example, will be able to obtain access to a system allowing him to use it on a selected set of hardware. Providing such software stacks to users is typically done in one of two ways: (a) the use of a pre-configured environment that is shared by all FG users, (b) the dynamic instantiation of a "private" environment controlled by the user (this environment can be shared based on the user's preferences and the policies of FG).

FG Interfaces. FG must be easily accessible to a variety of user communities. The user communities include FG system and application users, FG system developers, FG system administrators, and educational groups. To support this highly diverse user community, we need to employ command line tools, API's, and portals. Some of the communities will need their own specialized version of these tools to address their specific needs.

FG Stratosphere. FG provides functionality to monitor the hardware environment and the software executed on it. It also provides a sophisticated experiment management framework allowing users of FG to record and recreate experiments conducted in order to guarantee scientific reproducibility. To support this effort, planned experiments can be created through workflows, images can be stored and reused at a later time, and images can be generated based on simple descriptions in order to study the effect of different software stacks on applications.

FG RAIN - Runtime Adaptable INsertion Configurator. Dynamic provisioning is one of the central features of FG allowing users to instantiate images at runtime and execute their applications as part of these images. As already pointed out in the Software Section, multiple mechanisms exist to dynamically provision resources for the users need. RAIN will provide a comprehensive set of components to satisfy the different provisioning scenarios.

FG Security. An important component of FG is to deal with security. This includes the four A's: Authentication, Authorization, and Accounting. One of the major goals for security within FG is to enable single sign on for all users. Other tasks include the creation of vetted images to be used by the user community.

FG Support Software. FG relies on software that is provided by our team members to provide support for the entire FG user group. The FG project contains support to enhance this software in order to support the FG mission. Such software includes, but is not limited to Nimbus [7], Inca [8], Pegasus [9], ViNe [10], Vampir, and PAPI.

FG Applications. FG allows application users to try out the FG hardware and software in order to evaluate if a particular software environment is of interest and benefit for the application. Performance experiments can help assessing the validity of using a particular software stack, environment, or programming framework.

FG Partners. In future we intend to work with other partners beyond those funded by the original project. This includes participants in both academia and industry, depending on the needs of FG and the participating institutions.

For this paper, we will concentrate on a subset of architectural components that focus on the management of FutureGrid experiments. This includes in particular the organization of experiments, the generation of images, and the storage of images. Details of the dynamic provisioning are beyond the scope of this document and will be discussed in an additional paper that will be available shortly [11].

V. The FG Experiment Framework

Experiments are carried out using the scientific method to answer a question or investigate a problem [12]. As in physics, FG experiments typically contain one or more hypothesis that are supported by the experiment or disprove the hypothesis. They also include an experiment apparatus that is used to conduct the experiment. Proper recording of these activities not only allows the reproducibility of the experiment, but also the sharing of results within an interest group or the community. Moreover, an experiment apparatus can itself be a point of research or activities, that allow the creation of new experiments due to the sheer availability of the apparatus. This is a common model used in scientific discovery. For instance many astronomical discoveries would not have been possible without the invention of the telescope. FG provides such an elementary scientific instrument for system scientists. FG experiments require a sufficient description about the experiment so that a proper record, useful for the community, is preserved.

Activities within FutureGrid will be primarily experiment-based. These activities will be driven by steps that can be together classified as an experiment. Experiments may vary in complexity. They may include basic experiments, such as utilizing a particular pre-installed service and allowing a researcher debug an application interactively. They may also include more sophisticated experiments, such as instantiating a particular environment and running a pre-specified set of tasks on the environment. We envision that a direct outcome of having such a experiment-centric approach will be the creation of a collection of software images and experimental data that provides a reusable resource for application and computational sciences. FutureGrid will thus enable Grid and Cloud researchers to conveniently define, execute, and repeat application or middleware experiments within interacting software "stacks" that are under the control of the experimenter. It will also allow researchers to leverage from previous experiences of other experimenters in setting up and configuring experiments, hence creating a community of users. FutureGrid will support these pre-configured experiment environments with explicit default settings so that researchers can quickly select an appropriate pre-configured environment and use it in their specific scenario.

To better communicate the scope of the experiment related activities, we will first introduce a common set of terminology that we will use as part of this document. In the sections after that, we will introduce the scope of the Experiment Management System, describe the simple layered architecture,

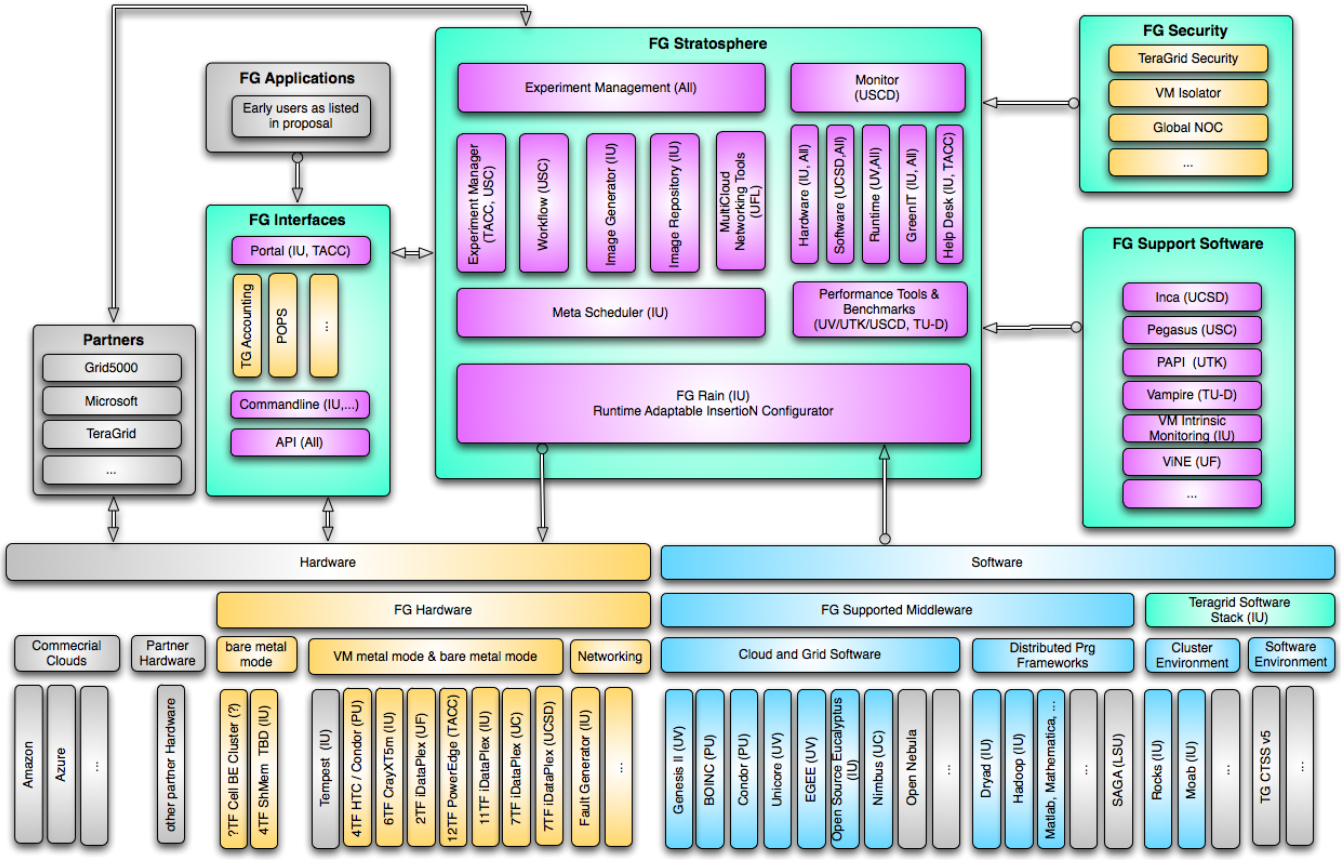

Fig. 3. FG Architecture Conceptual Overview

and identify tasks that need to be completed to successfully integrate such a service into FutureGrid.

In order for us to discuss the Experiment Management System we introduce the following simple terminology.

Experiment management. Experiment management refers to the ability of a test-bed user to define, initiate, and control a repeatable set of events designed to exercise some particular functionality, either in isolation or in aggregate.

Image. An image is series of bytes, namely a file, which can be loaded onto "bare metal" (real hardware) or on virtualized hardware using a hypervisor and will provide a complete and functional operating environment that the user can interact with. Only one image per node can be loaded directly to bare metal; multiple images can be loaded onto a hypervisor. Images do not extend across nodes but they can support multiple threads/cores. The bytes in an image encapsulates the state of persistent storage (e.g. a virtual hard disk, or a hard disk partition) that can be loaded onto bare metal/hypervisor. It is useful to distinguish appliances and generic images in that an appliance targets a specific application and a generic image only provides an environment for an experiment.

Generic Image. A generic image contains a basic operating system not targeted towards a specific use or application and as previously described, may be deployed on real hardware or on a hypervisor. Five hypervisors are of interest at present within FutureGrid, namely Xen, KVM, VirtualBox, ScaleMP, and OpenVZ. The two operating systems of interest for FG are Linux and Windows with a subset of interesting variants. For Linux these variants currently are Ubuntu, CentOS, and RHEL. On Windows we are interested in the newest generation of Windows software such as HPC server.

Appliances. An appliance is a generic image with additional application and/or middleware added that is configured and ready to use upon instantiation. However, initial calibration or tuning of the appliance will be often necessary before it can be used. This tuning step is hidden from the users of FG. Examples for appliances may include an image containing Gaussian, Matlab, Hadoop, MPI, or Oracle server. An appliance can present multiple applications and middleware to the user of the appliance.

Virtual Cluster. A virtual cluster is a collection of images with a virtual interconnect. Currently virtual interconnects are implemented by University of Florida via ViNE or GroupVPN. A virtual cluster maybe implemented on resources defined by the FG core services or the user can ask that a virtual cluster be assigned to a particular cluster.

Imaged Cluster. An imaged cluster is a collection of images deployed on hardware with a real interconnect. Typically

imaged clusters are achieved by defining a cluster and then specifying the images to be placed on each node. A traditional TeraGrid job would run on an Imaged Cluster. There is some need to define "allowed interference" that is the performance of an imaged cluster can depend on other users either using network of a given imaged cluster or even sharing a node within an imaged cluster. The key point to note here is that in contrast to a traditional HPC cluster, the Imaged cluster will be dynamically provisioned from the images, hence named the Imaged Cluster.

Account. For users to obtain access to FG, they need to apply for an account. An account must be part of at least one valid project to use FG. A user can be part of multiple projects, so there is no need for the user to have multiple accounts.

Project. A FG project is an elementary unit of request. A project serves as a convenient abstraction for the users to manage several experiments to fulfill one research goal. Different projects ar to be requested by the user to clearly distinguish the effort on the projects. A project provides general information about the requested resources, the group members that share the resources assigned to this project. It also describes the types of experiments that are anticipated to be conducted as part of the project execution. Each project has one or more experiments.

In order for a project to be granted data must be provided to support the project request including, but not limited to: a statement about scientific objectives, the anticipated projects to be executed, the team members conducting the research, the publications and presentations given based on the usage of FG (required to be added at a later time), the projects executed as part of the account activities and their experiments, and the projects broader impact.

Experiment. An experiment represents an elementary "execution unit". A project has a particular scientific goal in mind that may need the execution of one or more experiments. Such experiments may be organized in a tree or Direct Acyclic Graph (DAG) and contain other experiments. An experiment contains a number of important metadata: experiment session, the resource configuration , the resources used (apparatus), the images used, deployment specific attributes, the application used, the results of the experiments (typically files and data), and the expected duration of the experiment. An example of an experiment is running a Hadoop job as part of an academic class. If we view the class as a project, then each submitted student job could be viewed as an experiment.

Experiment Apparatus. Often it is desirable to conduct parameter studies or repetitive experiments with the same setup in regards to resources used. We refer to such a configuration as an "experiment apparatus". Such an apparatus allows the users to conveniently reuse the same setup without reconfiguration of the FutureGrid resources for different experiments.

Experiment Session. The experiment apparatus can be used for executing a number of experiments. In addition, the instantiation of experiments may require additional configuration in order to address runtime issues. Together the apparatus and the configuration parameters constitute an experiment session that can also be used for multiple experiments.

A. Requirements

In order to specify the requirements for the experiment management service, we have to consider the FG user communities. This includes:

FG experiment user, a general user who replicates experiments that have been provided by someone else.

FG experiment developer, user of FG who records his experiments for replicating and comparing results obtained in each instantiation of the experiment.

FG job user, a user that does not care about any of the issues and just wants to run jobs on FG.

FG experiment project manager, user that are manages the use of FG as part of projects or classes such as a teacher of a class of students. This user would create a set of experiments pertaining to the field of interest and classify these as a Project as described in section V-B.

FG system administrators, who can use information provided as part of the experiments to debug or improve the FG operations.

FG management, that is concerned with the use, reporting and auditing of FG activities.

In order to support these users, we need to identify certain functional requirements. These include but are not limited to the following functions that follow roughly a basic execution plan:

- Organize projects and experiments
- Provide a uniform structure so that organization of experiments is possible
- Annotate the experiments so they can be cataloged and shared (if desired)
- Annotate what the experiment is about
- Annotate which resources are being used
- Annotate which results are produced by the experiment
- Provide information about the nature of the projects and experiments to the FG management
- Provide a mechanism in which multiple users can easily collaborate as part of projects or even individual experiments
- Provision resources to conduct the experiments
- Execute an experiment
- Monitor the execution of experiments (irrespective of owner being individual, group, or management)
- Record all required information for replication of the experiment
- Reproduce the experiment with the help of the recorded information

B. Architecture Design Details

Next we focused on the design of the experiment management framework. As we organize projects and experiments we must chose an architectural design at an early stage that supports the functionality associated with it. The experiment management component is part of what we have called the FG stratosphere. In addition to the experiment management

```
ACCOUNTS
    USER+
      FIRSTNAME
      LASTNAME
      USER ID
      ...
PROJECTS
   PROJECT+
    APPLICATION INFORMATION
    DESCRIPTION
    USER ID
    ...
    PROJECT ADMINISTRATORS
        USER ID+
    USERS
        USER ID+

        EXPERIMENT+
            MANAGER
                USER ID+
            STAFF
                USER ID+
            APPARATUS+
                   ID
                   RESOURCE+
            EXPERIMENT+
                APPARATUS: ID
                DATE
                TIME
                JOB+
                SERVICE+
        PUBLICATIONS
            ARTICLE+
            REPORT+
            TALK+
            DEMO+
```

Fig. 4. FG experiment management data structure.

service, it also includes other core services, such as image management, dynamic provisioning abstractions (rain), and others. The relationship of the experiment management service towards other selected components is presented in Figure 3.

The experiment management service deals with management of data surrounding an experiment. Before we go into more detail of the architecture we would like to introduce the data structure that will allow us to deal with high level aspects of an experiment. In order to manage our experiments we envision the hierarchical data structure depicted in Figure 4. At this stage we list only the most important entities. A + behind the category means that the entity can occur one or more times.

The rationale behind the design of this data structure is best illustrated by a simple use case. Assume a teacher were to decide to use FutureGrid as part of his class resources to train the students on different aspects of Cloud computing. First the teacher has to apply for a project through the FG Web Site. The teacher may want to log into the resources, so also needs to apply for an account. Now the teacher would like to add a student to the project so a particular resource apparatus can be shared each week on which the student runs an experiment as an assignment. However, as the teacher is too busy dealing with the programming assignments, a teaching assistant is assigned to help. At the end each student is supposed to produce a class paper or project report, as well as give a presentation about their work. It is obvious that such a use case can easily be managed through our data structure. The Information related to the project is managed by the teacher. He delegates the addition of users to his system administrator that has a list of students in the class (in future additional convenience features may be available through the InCommon framework). The teaching assistant is added as staff member to the project and is allowed to generate new experiments, and experiment apparatus that are referred to within the experiments. The teacher decides to use the model where each week a new experiment is conducted, that contains for each student in the next level of the data structure hierarchy a placeholder for an experiment managed by a student for that week. Each student is in charge of uploading and managing its own experiment meta data. An apparatus that has been created by the teaching assistant can be shared amongst the students. This helps executing experiments on similar environments. At the end of the class paper and talks will be uploaded to the experiment management service.

The FG users, through convenient interfaces, are able to maintain the experiment management system and the data associated with it. The possible interfaces may include a Web portal, a command line tool, a REST interface, an API in Python (other scripting language interfaces could be derived from that).

C. Experiment Workflow Coordination

As the experiments may be hierarchical, we also provide convenient mechanisms to manage the metadata associated with complex hierarchical experiments. Experiments in form of a workflow are also planned. This includes the possibility of defining a workflow with common workflow services such as Pegasus. For the implementation of the workflow component we will utilize, on the lowest level, the dynamic provisioning functionalities from xCAT and Moab, with more generalized functionality provided by FG RAIN. The highest level of workflow coordination may be provided optionally through the use of workflow managers such as Pegasus.

It is important to recognize that the features provided by modern queuing systems have been significantly enhanced over the last decade and after we first envisioned the FG project. Thus we are leveraging functionality directly from other tools that are typically not provided in systems such as Globus, Condor, or even Pegasus.

As part of these new capabilities we are developing the *fg-rain* and *fg-submit* commands that will be ultimately helpful for providing many of the functionalities for provisioning and job submission. As Moab also provides cross site management, a future version of fg-rain and fg-submit may also provide the ability to integrate different resources from different sites into the same resource pool. In more detail we are developing:

- the specification of an easy to use command line tool named fg-experiment allowing the creation of sophisticated experiments and their coordination as workflow trees and DAGs in the context of FutureGrid,
- the development of a REST interface to interface with the functionality exposed as part of the fg-experiment command. This can be achieved while working together with IU, reusing their command line to REST services generation toolkit,
- the development of a python API with the namespace futureGrid and convenient classes and methods to provide the needed functionality,
- the development of educational material demonstrating the usefulness of the various user interfaces through tutorials and elaborate examples,
- the development of an experiment repository while leveraging the image repository, and
- an experiment management portal

additionally, the experiment management framework must be tightly integrated into the overall architecture of FG. Hence, we need to address the following essential requirements:

- the experiment management system must be integrated with the account management system,
- an experiment must be able to utilize the fg-rain, and fg-run, fg-hadoop, and other commands that are being developed in parallel,
- the workflow system must be controllable via command line and be accessible from shell scripts,
- the system must be easy to use and the prerequisites for its use must be simple,
- security must be considered while execution, but also sharing experiments,
- a python interface must exist, and
- the requirements on the system to be integrated into the experiment management harness must be minimal (e.g. we may not be able to expect that on each machine we must have Condor installed).

To support this integration we must have an experiment management information service that allows users to access information regarding accounts, projects, and experiments

Security requirements demand that the experiment use vetted images with appropriate privileges. Some privileges imply that the experiment be run with network isolation. The rules about privileges are less stringent for images built on virtual machines. The initial experiment request needs to be checked for being "reasonable" (e.g. consistent with project), and security issues. This motivates us to provide an image repository and an image generation service that can be utilized by the experiment management service.

D. Image Repository Service

The FG Image Repository service allows FG users to store the images that can either be owned by or shared among individual users or groups. Using the FG Image Repository service, users can query, store, share, upload, and register images, and choose them for dynamic provisioning. Most related Cloud service offerings provide a repository service. xCAT manages the images through Linux file structure and some table information; Nimbus uses the Linux file system and symbolic links to manage the images(though they are moving to a back end storage service schema similar to Amazon's S3 [13]); Amazon Web Services(AWS) [14], as well as its open source equivalence Eucalyptus [15], has the most sophisticated and well-defined functionality set and interface for an image repository service among these.

FutureGrid leverages efforts from xCAT, Nimbus, and Eucalyptus to provide an image repository for Cloud services. However, we have to remind ourselves that FG not only supports image usage within these frameworks, but even on a lower level. Hence, we need a unifying image repository that also integrates with our experiment management needs. None of the other systems provides this functionality. By developing an FG repository we can maintain specific data that assists performance monitoring and user/activity accounting. By introducing a customized image repository we will be able to choose an appropriate storage mechanism that is suitable for the FutureGrid platform

There are some key requirements and system constraints that have a significant impact on the architecture. These include:

- The image repository should be accessible by users through command line or a web portal. The security architecture imposed to the FG system ensures the security of this sub-system.
- The repository provides a unique interface to store various kinds of images for different systems, e.g., Nimbus, and Eucalyptus. The provisioning sub-system and the image generation service should be aware of this. That is, during image creation, appropriate attributes should be assigned according to future use; during system provisioning, the provisioning service will do the proper instantiation according to the image attributes.
- The repository could be a single point of failure and a bottleneck for performance. The storage mechanism should be distributed which provides higher performance and more reliability.
- To ease the development of the web portal client, REST services are preferred. Achieving security for the REST services needs to be considered along with the conformance of the whole FG security architecture.

The implementation of the distributed FG image repository contains a number of subcomponents that are depicted in Figure 5

E. Image Generator Service

The image generator service is the central component of the overall Image Management system. It is responsible for taking in user requirements about image size, type, and kind, and formatting a new image that, once vetted and stored, can be deployed on FG hardware. The image generator will start with a base image that is selected by the user. This image is specifically crafted by FG administrators to guarantee security

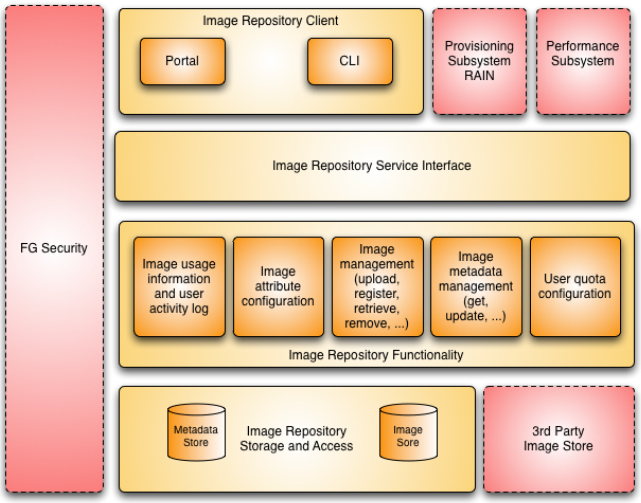

Fig. 5. Image Repository

and integration with the rest of FG. It is also designed to be the smallest file footprint possible, to minimize wait time and network traffic when deploying images. This image is next mounted and the software stack selected is deployed onto the system, along with any other files specified. The image generator then links the new image to BCFG2 [**?**] and submits it to the image repository.

BCFG2 retains another major component of the Image Management system. In fact, it does most of the management actions for all the VMs deployed throughout FG. BCFG2 itself is a critical tool to help system administrators produce a consistent, reproducible, and verifiable description of their environment, and offers visualization and reporting tools to aid in day-to-day administrative tasks. Within FG's BCFG2 deployment, we have a number of base deployment groups setup that correspond to the pre-supported OS types added by administrators to the Image Generator. From there, a given VM will be assigned another unique group which contains the software stack specified by the user. This allows for all software and files installed on the VM to be managed, updated, and verified by BCFG2. This group is created and defined by the image generator before initial deployment.

As described above, there are a number of base images that are supported within FutureGrid. These UNIX-based images represent the minimal installation possible within the OS itself. Because many of these image will be leveraged to provide platform-level services, there is no need to add extra packages and bloat to images, especially when the images are to be deployed and migrated throughout FG resources. The base OS is created as a separate .img file by FG administrators with the necessary BCFG2 client pre-installed along with any other monitoring software deemed fit by the FG Performance group.

Command line tools and an interface through a portal will simplify the generation of images allowing users to quickly generate and regenerate images.

The important issue to remember is that the image generation is integrated in a well defined process and allows for the generation of images which give access to users of FG either on an individual or a group basis. If user wants to share his image with other users, he can do so by either sharing how the image is to be generated, or by allowing access to a generated image with a particular group of FG users.

This image creation process is depicted in Figure 6. An image is generated by the user either via a command line or portal. He selects specific features of the image as needed including the target deployment selection such as OS, and hardware, as well as base software, FG software, Cloud software, user application software and other software. This creates then a *base image*. This base image is than deployed on a test server and updated and checked for security. The result is a deployable image on FG hardware. At time of deployment additional security updates are conducted. It is clear that the time between the creation of a deployable and deployed image has an impact on security. Images found insecure could still be deployed, but the network connectivity to such an image is restricted.

Our administrative staff will use the same process to create a base image for FG that can be modified by users through the process specified above. This makes it possible to provide different images on the lowest level and not just as part of a virtualized image deployment as part of IaaS frameworks. This is an extension to the Amazon model that only deals with deploying images in virtual machines.

As we expect that customized images will be contributed by the user community, we encourage a viral community model while emphasizing reuse of the images as they can be shared and distributed through our image repository.

VI. Related Research

The Grid community developed previously a number of systems dealing with workflow and experiment management systems. One such systems includes the Java CoG Kit that not only provided a sophisticated workflow system, but also defined very early on a prototype to manage job submissions to the Grid through an experiment management system [16], [17], [18]. Additionally, a follow up project of the Java CoG Kit known as Cyberaide [19] was providing a shell environment that allowed to conduct experiments within the shell using object abstractions. We hope at one point to reintroduce such a shell and object abstractions into our upcoming efforts. However, in contrast to these earlier efforts the use of an experiment in FG is more enhanced and includes besides the abstraction of a number of jobs submitted to a production Grid te staging and configuartion of software stacks.

Another important project is myExperiment [20], which allows the sharing of scientific workflows among its many users. This system has been successfully used. However, this system uses already deployed Grid infrastructure and does not take into account the specific needs of dealing with experiment management while integrating image management and image creation.

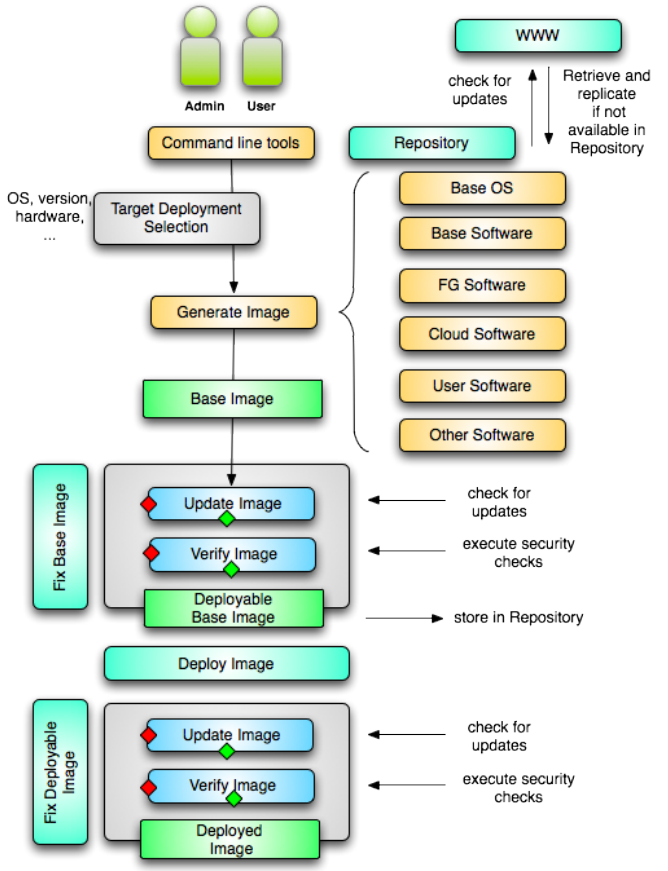

Fig. 6. Image Creation

VII. Conclusion

In this paper we have introduced the design of the FutureGrid experiment management framework. We focus on the concepts that are used to to manage experiments for preparing reusable and reproducible experiments. Essential to this management effort is how experiments are organized, they interface with user accounts, and how software stacks are integrated through the utilization of images. Such images are instantiated onto the FG resources through what we term raining. Once a stack is rained and properly configured it can be used to run experiments. The system is currently implemented. FutureGrid already provides today to a number of early users access to HPC, Grid and Cloud environments such as Nimbus, Eucalyptus, ViNe, Genesis II, and Unicore.

VIII. Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed." Partners in the FutureGrid project include U. Chicago, U. Florida, San Diego Supercomputer Center - UC San Diego, U. Southern California, U. Texas at Austin, U. Tennessee at Knoxville, U. of Virginia, Purdue I., and T-U. Dresden.

References

[1] "FutureGrid," Web Page, 2009. [Online]. Available: http://www.futuregrid.org

[2] *The Network Impairments device is Spirent XGEM*. [Online]. Available: http://www.spirent.com/Solutions-Directory/Impairments_GEM.aspx?oldtab=0&oldpg0=2

[3] *The FG Router/Switch is a Juniper EX8208*. [Online]. Available: http://www.juniper.net/us/en/products-services/switching/ex-series/ex8200/

[4] "xCAT Extreme Cloud Administration Toolkit." [Online]. Available: http://xcat.sourceforge.net/

[5] "bcfg2." [Online]. Available: http://www.adaptivecomputing.com/products/index.php

[6] "FutureGrid Hardware." [Online]. Available: http://www.futuregrid.org/hardware

[7] "Nimbus Project." [Online]. Available: http://www.nimbusproject.org/

[8] S. Smallen, K. Ericson, J. Hayes, and C. Olschanowsky, "User-level grid monitoring with Inca 2," in *Proceedings of the 2007 Workshop on Grid Monitoring (GMW'07)*. Monterey, CA: ACM, New York, 25 Jun. 2007.

[9] 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," *Scientific Programming Journal*, vol. 13, no. 3, pp. 219–237, 2005.

[10] M. Tsugawa and J. Fortes, "A virtual network (vine) architecture for grid computing," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, Apr 2006, p. 10 pp.

[11] G. von Laszewski and et. all., "FG Dynamic Provisioning," unpublished.

[12] Wikipedia, "Experiment," Web Page. [Online]. Available: http://en.wikipedia.org/wiki/Experiment

[13] "Amazon Simple Storage Service," Web Page. [Online]. Available: http://aws.amazon.com/s3/

[14] "Amazon Web Services." [Online]. Available: http://aws.amazon.com/

[15] "Eucalyptus Community." [Online]. Available: http://open.eucalyptus.com/

[16] G. von Laszewski, T. Trieu, P. Zimny, and D. Angulo, "The Java CoG Kit Experiment Manager," Argonne National Laboratory, Tech. Rep., Jun. 2005.

[17] G. von Laszewski, Gregor, T. Trieu, P. Zimny, and D. Angulo, "The Java CoG Kit Experiment Manager," in *GCE06 at SC06*, 2006.

[18] G. von Laszewski, "Java CoG Kit Workflow Concepts," *Journal of Grid Computing*, Jan. 2006, http://dx.doi.org/10.1007/s10723-005-9013-5.

[19] G. von Laszewski, A. Younge, X. He, K. Mahinthakumar, and L. Wang, "Experiment and Workflow Management Using Cyberaide Shell," in *4th International Workshop on Workflow Systems in e-Science (WSES 09) in conjunction with 9th IEEE International Symposium on Cluster Computing and the Grid*. IEEE, 2009. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/09-gridshell-ccgrid/vonLaszewski-ccgrid09-final.pdf

[20] D. D. Roure, C. Goble, S. Aleksejevs, J. B. Sean Bechhofer, D. Cruickshank, P. Fisher, N. Kollara, D. Michaelides, P. Missier, D. Newman, M. Ramsden, M. Roos, K. Wolstencroft, E. Zaluska, and J. Zhao, "The evolution of myexperiment," Website, 2010. [Online]. Available: http://eprints.ecs.soton.ac.uk/21458/1/myExpIEEEfinal.pdf