Enabling Workflow Repeatability with Virtualization Support *

Fan Jiang, Claris Castillo, Charles Schmitt, Anirban Mandal, Paul Ruth, Ilya Baldin
Renaissance Computing Institute
University of North Carolina Chapel Hill
Chapel Hill, NC, USA
{dcvan,claris,cschmitt,anirban,paul,ibaldin}@renci.org

ABSTRACT

The value of workflows to the scientific community spans over time and space. Not only results but also performance and resource consumption of a workflow need to be replayed over time and in varying environments. Achieving such repeatability in practice is challenging due to changes in software and infrastructure over time. In this work, we introduce a new abstraction that builds on the concept of virtual appliance to enable workflow repeatability. We have also developed a novel architecture to leverage this abstraction and realized it into a system implementation that supports a popular workflow management system and builds on a federated in-production environment. We demonstrate the effectiveness of our approach by examining various aspects of workflow repeatability. Our results show that workflows can be replayed with 2% fidelity when considering their walltime as performance metric.

Keywords

workflow; repeatability; virtual appliance;

1. INTRODUCTION

Conceptually, workflows are collaborative artifacts. They encode scientific methods that allow for preservation, replay and reuse, which are key ingredients for scientific collaboration. As workflows need to be replayed over time and in different environments, enabling workflow repeatability is of great importance to the scientific community. Others in the community [4][5][19] have developed approaches to effectively capturing workflow properties for re-execution. However, these approaches aim at reproducing the same results across workflow executions and typically encapsulate data and code of a workflow, but seldom include information about the infrastructure and its resources. Absence of this information hinders workflow execution due to issues such as availability of infrastructure, compatibility issues caused by software updates, invalid configuration due to changes in runtime, etc.

In general, the problem of computational repeatability has been recognized by the scientific community [10][11][21]. Re-execution of workflows presents unique challenges as compared to traditional compute paradigms. These challenges stem from the fact that their execution often span over multiple geographically distributed resources. With the adoption of federated Cloud environments, such as ExoGENI [9] and FutureGrid [22], the problem is exacerbated further since resources availability is subjected to the policies of the federated sites.

Moreover, performance is also a crucial aspect for workflow replay. Note that in this paper we use "re-execute" and "replay" interchangeably when referring to repetition of workflow execution. There are two aspects of workflow performance we are concerned with in this work. Firstly, a workflow is expected to exhibit comparable performance with equivalent resources across time. Secondly, control over resource and infrastructure configuration should yield predictable workflow performance to scientists who want to re-execute workflows under different resource and infrastructure constraints.

Virtual machines (VMs) have been proposed as the *only* mechanisms for ensuring recomputability of experiments and the basic model of repeatable research [11] [21]. More recently, virtual appliances have been adopted as a solutions to reducing development and distribution costs [20]. Virtual appliances are pre-built, self-contained and pre-configured software solution comprising of multiple VMs that are packaged, updated and managed as a unit. We subscribe to this vision and argue that to enable workflow re-execution this concept has to go beyond the virtualization of computation - in the form of VMs - to include the execution environment (i.e. the network, storage and data sources) whenever possible.

To this end we introduce a new resource abstraction, called *workflow virtual appliance* (WVA). A WVA is a virtual appliance that encodes every piece of the puzzle for deploying a workflow, including the software, data and its execution environment. To achieve this it leverages two state-of-the-art technologies: (1) virtualization technology to encapsulate software stack and its configuration and (2) virtual infrastructure abstraction, namely *slice*, introduced in the context of Networked-Infrastructure-As-A-Service (NIaaS) [9], which provides the resource description of mutually isolated pieces of networked virtual infrastructure carved out from multiple Cloud and network transit providers.

A WVA is realized using a pre-integrated, self-contained object (i.e. file) that contains information on (1) the full suite of software (e.g. workflow management software) and the configuration required for executing a given workflow, and (2) the *slice*, description of its virtualized environment (e.g. VM images, resource capacities, etc.). This information is in the form of a machine-readable object format that can be instantiated (i.e., deployed and replayed) in a NIaaS Cloud environment in the same manner a VM image is used to instantiate a VM. A WVA file contains static content that

*SC15, November 15-20, 2015, Austin, Texas, USA 978-1-4799-5500-8/14/$31.00 ©2015 IEEE

WORKS '15 November 15–20, 2015, Austin, TX, USA

DOI: 10.475/2822332.2822340

can be transmitted across the network for sharing, or registered with some data store for later retrieval. This effectively addresses the basic problems of workflow sharing and workflow decay [24] that the scientific community faces today. Moreover, coupling the application with the virtual infrastructure into a virtual appliance enables the development of novel capabilities such as checkpointing workflow environments.

We have realized these ideas into the architecture of a Cloud-based system, named *WRAP* (Workflow Repeat APpliance), that builds around the concept of WVA. *WRAP* consists of a multi-modular middleware that presents users with relevant interfaces for composing, modifying and storing WVA objects as well as deploying (executing) and snapshotting workflow environments. In *WRAP*, WVA objects are the units of deployment and therefore, a scientist only needs to provide a correct WVA object for executing a workflow. *WRAP* follows a Service Oriented Architecture (SOA) model and is composed of a set of services that together automate the deployment of virtual infrastructure on the NIaaS and the execution and management of the workflow. In addition, it builds on state-of-the-art virtualization technology to allow scientists to snapshot the state of their workflows and hosting infrastructure into a completely new WVA object for sharing or re-execution in the future.

We have evaluated the feasibility and effectiveness of *WRAP* in enhancing workflow repeatability against real workflows from the genomic science and astronomy domains. Our results demonstrate that *WRAP* can significantly reduce the complexity experienced by scientists in sharing and replaying workflows. More importantly, we also show that workflows can be replayed with high performance fidelity even for significant infrastructure modifications in a shared federated Cloud environment in production. For example, our workflows exhibit performance variance of less than 1 % and 3% for changes to the execution sites and execution times (spanning over several weeks), respectively.

In summary, our contributions in this paper are two-fold:

- We introduce a new abstraction for workflows (WVA) which builds on the concept of *virtual appliance* and encapsulates the entire workflow execution environment, including both workflow and infrastructure information, to enable workflow repeatability and improve scientific productivity. To the best of our knowledge, this is the first work to explore the use of virtual appliance for encapsulating execution environment including both networked infrastructure and workflow application.

- We present a novel software architecture to enable workflow repeatability on IaaS environments using WVA abstraction. The architecture is realized into a system implementation that supports a popular workflow management system (Pegasus [8]) and builds on an in-production NIaaS environment(ExoGENI [9]). We further evaluate the effectiveness and efficiency of our system against real workloads.

2. RELATED WORK

Seminal work in computational repeatability in a broad context includes [11], [21] and [10]. In [11] the author proposes VMs as the only credible mechanisms for replaying computation. Furthermore, the author propses a set of standard practices for using VMs to replicate experiments in Computer Science. In [21] the authors explore a repeatable model consisting of performing one's experiment within a VM hosted by some Cloud providers. When the experiment is complete, the experimenter snapshot the VM and make it public for others to use. *WRAP* goes beyond this model in that it captures complete networked infrastructure consisting of multiple VMs, storage, data sources and networks spanning over multiple network provides by relying on existing appliances (*slice*).

In [18] the authors present one of the seminal works in workflow conservation. They introduce *pack* as the unit of sharing. A *pack* is similar to a WVA in that it consists of a package of components that make up an experiment. However, a *pack* only includes content pertaining to ownership, intellectual property and provenance of the workflow. Hence, it is not deployable. This is in contrast to a WVA object which is self-contained and can be deployed on a NIaaS.

More recently, in [19] the authors propose a semantic-based approach to produce workflow specifications that can be used for re-execution. The specification includes a semantic description for both the workflow and the resources. We believe that providing such specification is not a trivial task for domain scientists with limited computing skills thus hindering the adoption of this approach. Previous work done by the authors of [12] is also relevant in that it consists of a semantic approach for describing workflow components. This work however does not consider resources or infrastructure.

3. BACKGROUND AND MOTIVATION

In this section, we introduce three state-of-the-art technologies used in our work. We further motivate our work by describing step-by-step the process to replay a workflow using these technologies. We hope that by doing this readers without hands-on experience executing workflows can develop a strong intuition of the complexity behind replaying a workflow across time and space.

3.1 Used Technologies

3.1.1 HTCondor

HTCondor is a compute framework that enables High Throughput Computing (HTC) on large collections over distributed computing resources [13]. It implements a "master/slave" paradigm to take advantage of parallelism and has been adopted in experiments with scientific workflows. A typical HTCondor deployment is composed of a master node and a number of slave nodes. The master performs job scheduling. It matches computation jobs to slave nodes based on job requirements and the characteristics of the nodes. The slave nodes are responsible for making progress on the jobs. Classified Advertisements (*classads*) are exchanged between master and slave nodes to schedule jobs and report current state and resource availability of each slave node.

3.1.2 Pegasus

The Pegasus Workflow Management System [8] has been used by scientists in many domains to execute large-scale computational workflows on a variety of cyberinfrastructure, ranging from local desktops to campus clusters, grids, and commercial and academic Clouds.

The key idea behind Pegasus is the separation of the workflow description from the description of the execution environment, which results in workflow portability, and the ability for the workflow management system to make performance focused decisions at "planning time" and at "runtime". This separation is the key differentiator between Pegasus and the other workflow management systems such as Kepler [3] and Taverna [23]. In fact, both Kepler and Taverna have the SOA; hence, the resource model maps computation directly to web services. A Pegasus workflow on the other hand can run on any resource platform as long as the Pegasus software stack –including HTCondor– is installed, configured

and available. This resource model allows for using virtual infrastructure which is in full alignment with the concept of virtual appliance which serves as the basis of our work.

3.1.3 ExoGENI

The ExoGENI testbed, as shown in Figure 1, consists of Cloud sites (*racks*) on host campuses, linked with national research networks through programmable exchange points. Networked Cloud infrastructures link distributed resources into connected arrangements, *slices*, targeted at solving a specific problem. This *slice* abstraction is central to provide mutually isolated pieces of networked virtual infrastructure built to order the guest applications. Compute and storage resources are obtained from private Clouds at the infrastructure's edge. Network resources are obtained from edge providers and national fabrics using traditional VLAN-based switching and OpenFlow. Using ORCA [7] (Open Resource Control Architecture) control framework software, ExoGENI offers a powerful unified hosting platform for deeply networked, multi-domain, multi-site Cloud applications. What sets ExoGENI apart from most Cloud systems is its ability to allocate bandwidth-provisioned dedicated layer-2 private networks between compute resources across Cloud sites. These dedicated layer-2 networks enable fast transfer of data files between computational tasks.

In the context of workflows, ExoGENI has been used to host scientific workflows from multiple domain sciences and important efforts are currently enhancing its control framework to better support them [14].

3.2 Motivation

To further motivate our work, let us describe the process to compose, run and share a Pegasus workflow on a NIaaS such as ExoGENI. First, the scientist requests the deployment of a *slice* consisting of a network of VMs potentially allocated across multiple Cloud sites. Figure 5 depicts one *slice* used in our evaluation. Once the VMs are up and running, the user logins to install and configure Pegasus/HTCondor software.

Following the procurement and configuration of the infrastructure, the user creates a Pegasus workflow. To do this, she logins into the master VM node and composes an abstract representation of the workflow using a Pegasus high-level specification language called DAX. A workflow abstract consists of computation tasks (programs) and data. These abstract artifacts are represented as logical names. In order to execute the workflow, its abstract representation is converted into an executable representation consumable by HTCondor called Dagman. This conversion requires a set of pre-populated catalogs with the mappings of each logical entity (e.g. dataset) to its corresponding physical location (e.g. file paths, URLs). For instance, for every computation task the scientist has to provide the physical location of the application, e.g. OS requirements, environment variables and any other configuration needed by HTCondor to identify a suitable compute host.

In order to replay this workflow in a different environment a user has to procure a virtual environment that follows closely the original one. This step includes the deployment and configuration of virtual infrastructure as well as the installation and configuration of the software suite. The process of replicating the same environment involves substantial interaction with IT staff. Furthermore, all the application-specific information, including the DAX file, programs and data, needs to be shared. Finally, catalogs must be populated in advance with a mapping of logical to physical entities that reflects the new infrastructure.

It is worth noticing that in the case of workflow management systems such as Kepler [3] and Taverna [23] users face even higher technical barriers for sharing and replaying workflows. This is due to the fact that both systems rely on third-party services for running computation.

Furthermore, changes in the compute environment are the norm. A minor change may have tremendous effect on workflow performance. For a scientist, in order to know the effect of these changes in advance, she needs in-depth knowledge of the workload. For instance, a data-intensive workflow may exhibit low performance when running on VMs with larger cpu allocation but marginally smaller network allocation. This is particularly important as users aim to maximize their return on investment (ROI) in a Cloud business model and have higher Quality of Service (QoS) demands (e.g., meeting deadlines). As part of our research agenda we are concerned with developing mechanisms that allow scientists to predict workflow performance in the presence of changes in the virtual environment. In Section 5 we demonstrate that *WRAP* is a key enabler to attain this goal by encapsulating both virtual infrastructure and application into one single appliance and allowing users to modify resource capacities and capabilities.

4. WRAP ENVIRONMENT

4.1 Workflow Virtual Appliance (WVA)

A WVA is a pre-built self-contained object implemented as a file that describes the full stack of hardware and software required for a workflow ensemble. A workflow ensemble consists of the execution of multiple instances of the same workflow with different data input. Listing 1 shows an example of a WVA describing an ensemble of two genomic analysis workflow instances.

A WVA is composed of two major fields - *ensemble* and *infrastructure* - to describe the application and infrastructure used in a specific workflow.

The *ensemble* field contains all the application information needed for configuring the workflow management system and instantiating workflow instances. The workflow management system is specified in the *platform* field. The *config* field includes configuration properties specific to the workflow management system. The workflow is described in the *workflow* field with a specific schema consumable by the workflow management system. For instance, Pegasus uses DAG. Discussion on workflow composition in Pegasus is beyond the scope of this paper. We assume that the scientist possesses knowledge in developing workflow descriptions for a specific workflow management system she chooses to use. Workflow instances are included in the *workflow-instances* field. They differ from each other mainly in their input datasets, which are presented with URIs in the *input* field. Output data files of each workflow instance will be transferred to the location specified in the *output* field. The performance measurements are aggregated for each workflow instance as shown in the *performance* field of each instance. Besides the end-to-end execution time (*walltime*) shown in 1, we also collect CPU and memory utilization, disk *read*/*write* rates and network *send*/*recv* rates in current implementation.

The virtual infrastructure, referred here as a *slice* to follow ExoGENI terminology is described in the *infrastructure* field. ExoGENI encodes resource descriptions using the Netowrk Description Language (NDL) language [2]. In this field we include all the parameters that are relevant to the workflow including the VM images, resource capacities, storage, etc. The *image* field contains a reference to a VM image with pre-installed executables. *Master* and *slave* nodes are declared individually with VM *type* and *links* connecting them. A VM *type* maps to a specific combination of computing resources. In this example we use ExoGENI as the testbed and the VM *type* is one of the following types: small (1 core, 1GB

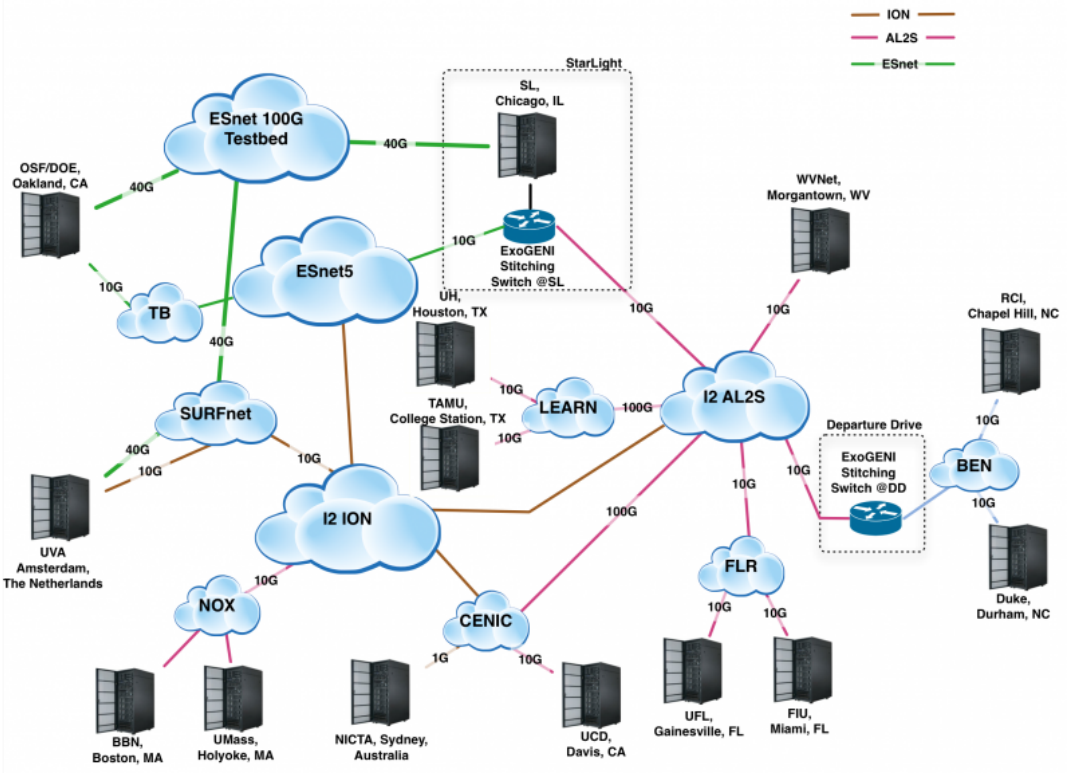


Figure 1: ExoGENI

RAM and 10GB disk), medium (1 core, 3GB RAM and 25GB disk), large (2 cores, 6GB RAM and 50GB disk) and extra large (4 cores, 12GB RAM and 75GB disk). ExoGENI supports network attached storage. The capacity of the storage (GB) and its corresponding network link (Mb/s) are defined in the *size* and *links* fields, respectively. In NDL, links can be referenced in entities, entities can be interconnected and a infrastructure topology shown in Figure 5 can be described.

```
{
 "id":"...",
 "name":"Genomic analysis ensemble",
 "ensemble":{
   "platform":"pegasus",
   "config":{...},
   "workflow":{
     "executables":["bwa",...],
     "jobs":[
        {"id":"1","cmd":"bwa aln","input":[1],"output":[3]},
        {"id":"2","cmd":"bwa aln","input":[2],"output":[4]},
       ...
     ],
     "dependencies":{"1":["2"],"2":["3"],...},
     ...
   },
   "workflow-instances":[
     {
      "id":0,
      "input":[
        "irods:///user/data/SRR824936_1.filt.fastq",
        ...
      ],
      "output":"scp://172.16.1.1:22/output/",
      "performance":{
        "walltime":{"max":17010,"min":16540,"avg":16680},
        ...
     }},
     ...
   ]},
 "infrastructure":{
   "images":[
     {"id":"g0","url":"http://wva.renci.org/geno/image"},
     ...
   ],
   "nodes":[
     {"id":"m0","site":"osf","image":"g0","role":"master",...},
     {"id":"db0","size":30,"site":"osf","role":"storage",...},
     {"id":"s0","site":"rci","image":"g0","role":"slave",...},
     ...
   ],
   "links":[
        {"id":"l1","bandwidth":100,"source":"m0","target":"db0
             "},
        {"id":"l2","bandwidth":500,"source":"m0","target":"s0
             "},
        ...
   ],
   "reservation":24,
 }
}
```

Listing 1: WVA example

It is worth noting that most fields in a WVA file except those pertaining to performance are immutable, because any change of these field will result in a different workflow execution environment. Hence, users can create newer version of a WVA object. Thus, *WRAP* provides provenance capabilities on WVA objects.

4.2 WRAP Architecture

WRAP is a platform that manages the construction, monitoring and replication of virtualized environments built upon WVAs. In *WRAP* , WVA is the unit of deployment and execution. It follows the Service Oriented Architecture (SOA) and integrates a set of RESTful services where each service is self-contained and exposes a well-defined API to the others. The services are optionally unified with a frontend to increase the usability of the system. Figure 2 depicts the key components of our system.

4.2.1 WVA Catalog

The *WVA Catalog* stores WVA objects that can be retrieved for workflow re-execution. It is backed with MongoDB [15] and presents a CRUD API for creating, reading, updating and deleting WVAs.

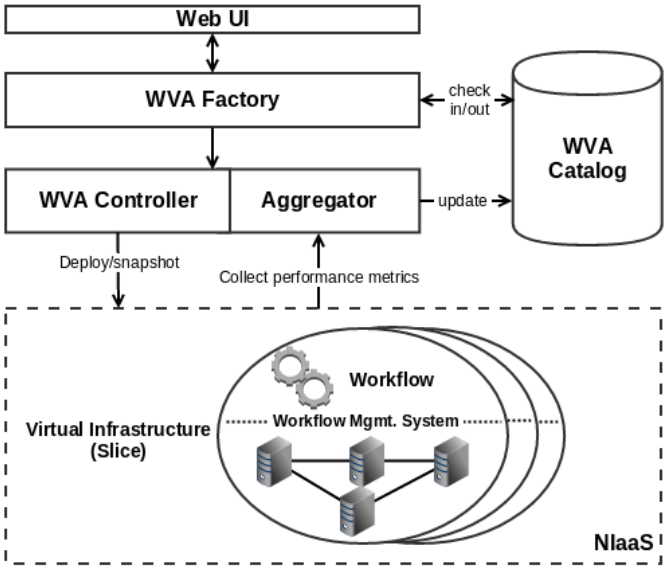


Figure 2: WRAP architecture

Users can retrieve an existing WVA for reconstructing an WVA environment or delete obsolete WVAs through the API. However, users are restricted from creating new WVAs directly on the *WVA Catalog*. Instead, the *WVA Factory*, which will be introduced later in this Section, will perform the operation on behalf of the user. By doing this we ensure the correcness of WVAs stored in the *WVA Catalog*. The *WVA Factory* is the only component permitted to update an active WVA.

4.2.2 WVA Controller

The *WVA Controller* is the main actuator entity of the system. It translates user requests into requests consumable by the NIaaS service provider, submits the translated requests to the NIaaS service provider for fulfillment and notifies the user of its status. Among the various user requests handled by the *WVA Controller*, we discuss two major types of request in detail: a *deployment* and *snapshot* request.

- The *deployment request* includes almost the WVA in its totality. The *WVA Controller* reassemble fields extracted from the WVA object into a NDL request issued to the NIaaS for procurement. By the same token, the *WVA Controller* compose a DAG file from the information included in the *workflow* and *workflow-instances* fields in a format compatible with the workflow management software. A configuration file for the workflow management software is also created using properties extracted from the *config* field. We use VM boot scripts to initialize the newly established virtualized environment. Boot scripts are scripts that encode initialization actions that must be executed immediately after the VM boots. The boot script configures the workflow management software with the configuration file and launches the workflow instances using the DAG files on the master node. It also starts the monitoring processes, which automatically collects performance measurements from the virtualized environment.

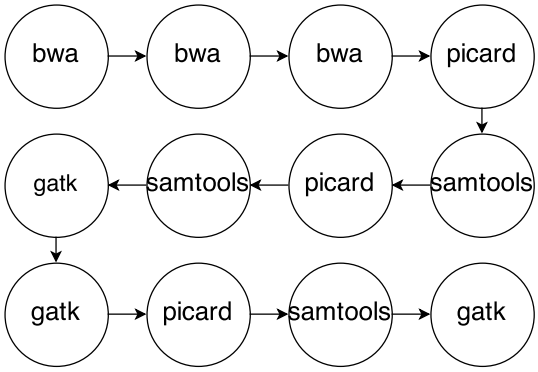
- The *snapshot request* is issued by a user to replicate an active virtualized environment entirely with its current state at a given point in time. The replication is enabled by the VM snapshotting feature available in modern virtualization technology, which allows users to restore a VM to a particular state by providing change log for the virtual disk. Workflow management software suites like Pegasus also have workflow recovery mechanism that allows workflow instances to be suspended and restored to the previous state later on. The snapshot of a virtualized environment occurs at both application and infrastructure levels. At the application level, workflow instances are suspended and their states are persisted immediately on receipt of the snapshot request. At infrastructure level, current virtual infrastructure is captured. The *WVA Controller* requests the NIaaS service provider to take a snapshot of the master node on which states of hosted workflow instances are stored. Besides, all the resource parameters of the virtual infrastructure, including those dynamically allocated such as internal IP addresses, are captured in order to keep the replicated environment consistent with the original one from all aspects. A new NDL request is created with new image reference to the VM snapshot and captured resource parameters of the virtual infrastructure. The new NDL request together with the old WVA are sent to the *WVA Factory* to create a new WVA.

4.2.3 WVA Factory

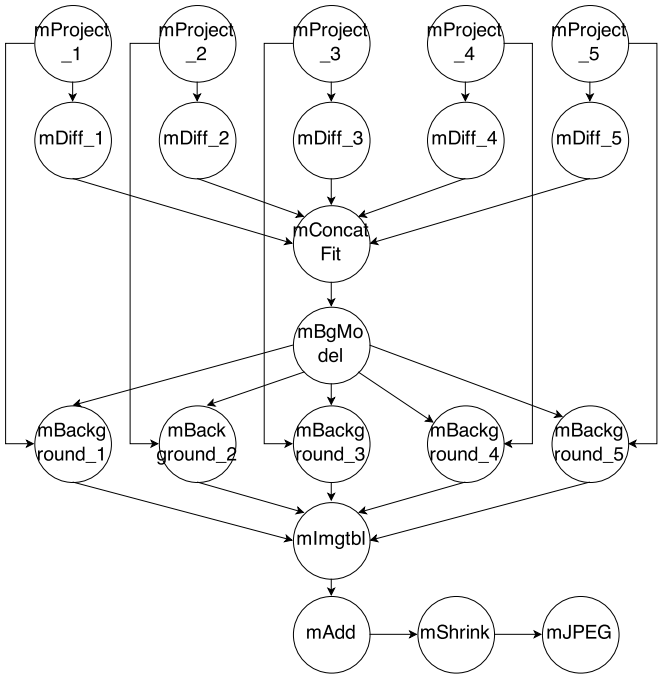
The role of the *WVA Factory* is to compose new WVAs and update existing WVAs with performance measurements received from active WVAs. New WVA objects are validated before they are stored in the *WVA Catalog*. As of writing of this paper only syntax checking is performed to validate a WVA object. If the validation fails, it notifies the user about the failure with details about the anomaly found. Valid WVA objects are stored in the *WVA Catalog*. There is also an *aggregator* process running on the *WVA Factory*, which is a part of the *monitoring framework* used in the system. We will introduce the process in detail later.

4.2.4 Monitoring Framework

The *monitoring framework* consists of an *aggregator* process residing in the *WVA Factory* and a set of *collector* processes distributed on both master and slave node VMs. The *aggregator* process listens on messages containing performance measurements data from the *collector* processes. Upon completion of an ensemble, the *aggregate* process aggregates the measurements and updates the corresponding WVA in the *WVA Catalog*. The *collector* processes collect performance measurements from the VMs in the virtual infrastructure. The collector process running on the mas-



(a) Genomic



(b) Montage

Figure 3: Workflows used in evaluation

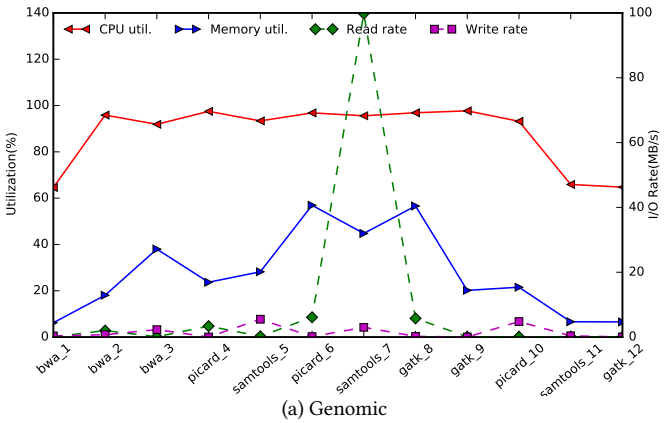
ter node only captures workflow-specific measurements, including walltime. It reports the measurements to the *aggregator* every 30 seconds. The *collector* processes on the slave node VMs focus on system measurements, including CPU and memory utilization, disk I/O rate and network I/O rate. Both the master and the collector processes are implemented in Python. The process on a slave node VM starts monitoring only when a job is running in the VM. Measurements are collected and reported every 2 seconds to match the time granularity of very short jobs. The *aggregator* and the *collector* processes communicate with each other using asynchronous messaging implemented using RabbitMQ [17] because of its scalability and performance.

5. EVALUATION

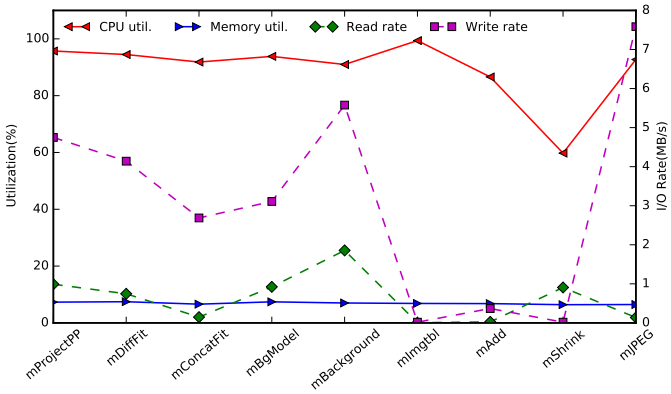
We have performed an evaluation with real workloads to validate *WRAP* 's ability to enable workflow repeatability. We also show how *WRAP* can facilitate performance predictability. In this section, we also discuss our results.

5.1 Experiments

We perform a set of experiments to investigate the level of fidelity of execution achieved when the workflows are executed at



(a) Genomic



(b) Montage

Figure 4: Resource demand for genomic and Montage

different time and using resources from different ExoGENI sites. We also examine the impact on performance of three factors: VM capacity, number of VMs and capacity of network links.

We consider user-centric and system-centric performance metrics to drive our evaluation. We use *walltime* as user-centric performance metric and CPU utilization, memory utilization and disk I/O rates as system-centric metrics. The performance measurements are collected and aggregated using the monitoring framework built in *WRAP* .

Our experiments are hosted in ExoGENI. By doing this we are able to specify resource capacities of the hosting virtual infrastructures including VMs (CPU, memory), storage and network. More importantly, ExoGENI allows us to define network connectivity between compute nodes (VMs) thus. VM types currently available in ExoGENI are shown in Table 1. We also specify the bandwidth allocated to every network link in the virtual infrastructures. In addition, the experiments were run over 10 of ExoGENI's worldwide distributed sites shown in Figure 1, thus stressing the ability of *WRAP* to replay workflows at scale and under high levels of substrate heterogeneity.

Name	Cores	RAM(GB)	Disk(GB)
Small	1	1	10
Medium	1	3	25
Large	2	6	50
XLarge	4	12	75

Table 1: ExoGENI Resource Types

To drive our experiments we use two real workflows: The exomic alignment workflow [16] (Figure 3a) and the Montage [1] workflow (Figure 3b). The exomic alignment workflow, referred to as genomic workflow, is based on a production workflow for clin-

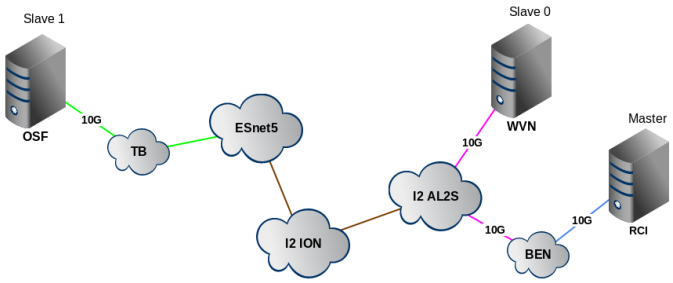


Figure 5: Multinode topology

ical genomics at University of North Carolina at Chapel Hill. It is composed of a pipeline of 12 jobs as depicted in Figure 3a. It has an input and an output of 352MB of exome sequence data and 1.1GB of aligned exomes, respectively. It also generates 1.4GB of intermediate data which have to be transferred over the network during its execution. The Montage workflow is an astronomical workflow created by NASA/IPAC for generating one square degree 2MASS J-band mosaic centered on M17 [6] [1]. It consists of a combination of parallel and pipelined jobs as depicted in Figure 3b. The Montage workflow has an input and an output of 4.9GB of Flexible Image Transport System (FITS) images and 7.4GB of mosaic images, respectively. The amount of intermediate data it generates amounts to 17GB. Figure 4 shows the resource demand for CPU, memory and I/O for both workflows. As we can observe, the Montage workflow is more data-intensive than the genomic workflow as it involves more data transfers.

We use Pegasus to manage workflows, which relies on HTCondor as its compute platform as described in Section 3. We use two virtual infrastructure topologies, *standalone* and *multinode* (Figure 5), to accommodate different workflows requirements. The *standalone* infrastructure is composed of one single VM, which includes all the software components required for workflow execution and runs as both a master and a slave node. Workflows with serial jobs such as the genomic workflow can benefit from this infrastructure because of its simplicity, efficiency and low cost. The *multinode* infrastructure is designed for workflows with parallel jobs, such as Montage. In this infrastructure, there is a master node VM dedicated for workflow management and multiple interconnected slave node VMs. Workflow ensembles with pipelined workflows can also take advantage of this infrastructure to centralize management of individual workflow instances on the master node, reducing resource overhead on workflow management. To study the impact of network capacity on workflow performance, we have the slave nodes stage intermediate data back to the master node in order to saturate the network links.

5.2 Repeatability

We want to demonstrate via experimentation that the adoption of *WRAP* in a virtualized federated environment such as ExoGENI enables the re-execution of workflows with high fidelity in the presence of major changes to the underlying infrastructure (e.g., hosting virtual infrastructure in different sites). We use *walltime* to measure the fidelity of the workflow execution.

The experiments were run against both workflows. The genomic workflows and the Montage workflows were hosted in a *standalone* infrastructure with a large VM and a *multinode* infrastructure with 5 large VMs interconnected with 100Mb/s network links, respectively.

We perform the experiments in two groups. In the first group, we run the workflow within one single site. This case is representative of a common scenario where computation is run on a local site. We execute the workflows on multiple sites and compare their performance. As shown in Figure 6, the walltime remains very stable in response to changes in sites for both workflows. The walltime standard deviation is 13.57 and 2.83 minutes out of 188.88 and 103.76 minutes on average for the genomic and Montage workflow, respectively.

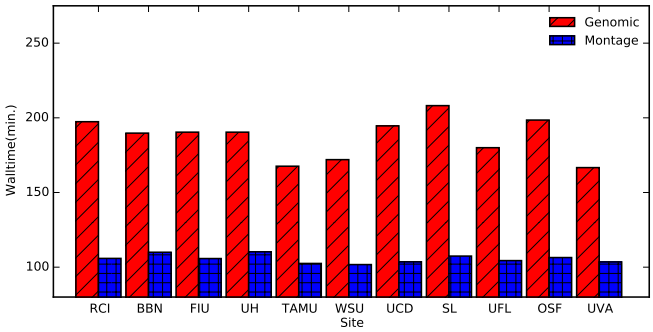


Figure 6: Walltime on varying ExoGENI sites

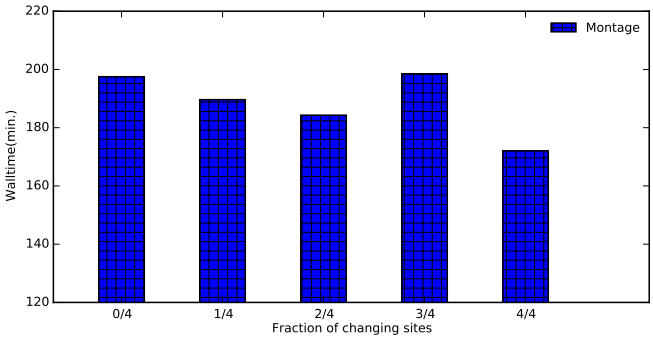
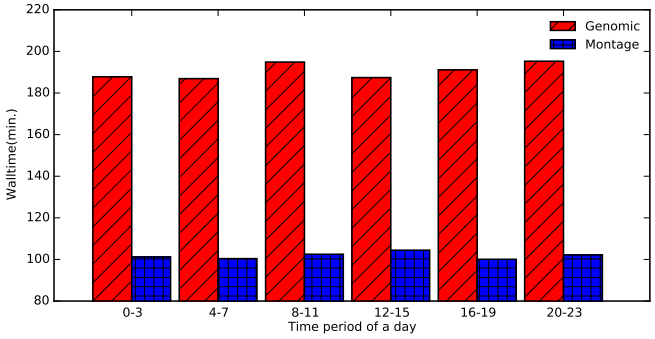


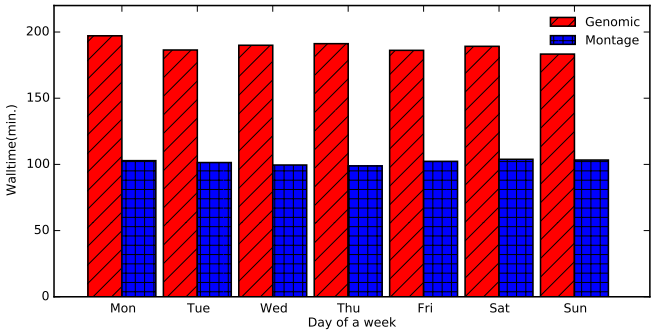
Figure 7: Walltime with varying number of changed sites

In the second group we consider the scenario where the workflow is hosted in a fully distributed platform. We deploy 4 VMs on 4 different sites; VMs are interconnected with 100Mb/s network link. To conduct this experiment we start by running the workflow on N sites. We then change one, two, three,..., N sites on the original setup and run the workflow on each of these new configurations. For example in Figure 7 $1/4$ and $3/4$ in the X-axis denotes that 1 and 3 sites are changed from the original infrastructure, respectively. The initial deployment spans over sites located at UFL, UCD, TAMU and OSF. The experiment has been only conducted against Montage workflow because the genomic workflow is rarely executed over multiple sites due to its serial structure. Figure 7 depicts the results. The selection of the site to change and its replacement was performed randomly. As we can observe, the walltime varies more as compared to the intra-site experiments. This result follows intuition since we introduce a higher level of substrate heterogeneity when multiple sites were involved including different network providers and datacenter hardware. However, the performance variation is acceptable with the walltime standard deviation at 10.87 out of 189.45 minutes on average.

We also investigate the repeatability achieved when executing the workflow at different times. This evaluation stresses the capabilities of *WRAP* in a federated Cloud under different load levels. We launch workflows at different times of a day and different days of a week during January 25th and February 23rd, 2015. The experiments with the genomic workflow are run on the UH site and those with Montage workflow are performed on the WSU site. Figure 8a and Figure 8b show the results. When the workflows are run at different times of the day, the walltime standard deviation is 3.79 and 1.89 out of 190.39 and 101.74 minutes for the genomic and Montage workflow, respectively. For the experiment with varying days of a week we observe slightly higher standard deviation for



(a) Walltime at different times of a day



(b) Walltime on different days of a week

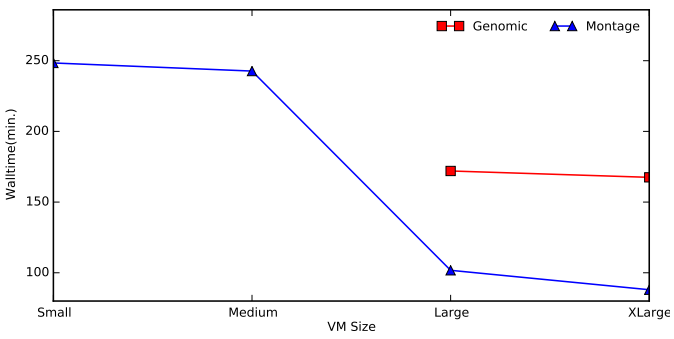
Figure 8: Walltime at varying time of execution

both the genomic and the Montage workflow being 4.45 and 2.02 minutes, respectively. Our result demonstrate that *WRAP* achieves fidelity in workflow execution by effectively leveraging the performance isolation achieved by modern virtualization technologies in ExoGENI. This observation demonstrates that the use of virtual appliances and virtualization technologies are suitable to address the problem of workflow repeatability addressed in this work.

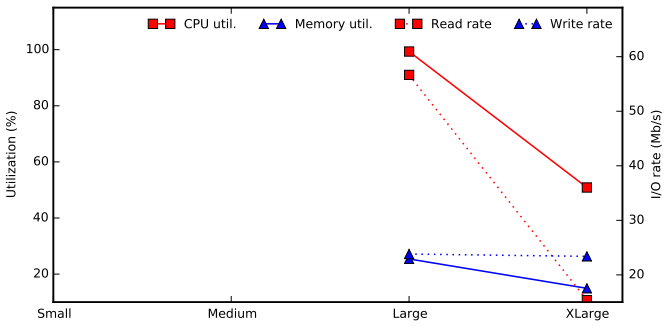
5.3 Performance predictability

As described in Section 4, a WVA object includes information pertaining to the infrastructure targeted for a given workflow including capacities and capabilities of resources. A scientist can modify this information in a WVA file to influence the performance of a workflow and satisfy user requirements (e.g., meet deadlines, reduce resource allocation in view of high resource demands). Being able to control workflow performance with reasonable levels of predictability is due to both resource management of virtual resources in ExoGENI (e.g., performance isolation) and the proper leverage of this feature by *WRAP*. To further investigate this goal we benchmark the two workflows as a function of *VM capacity*, *number of VMs* and *network capacity of links between VMs*.

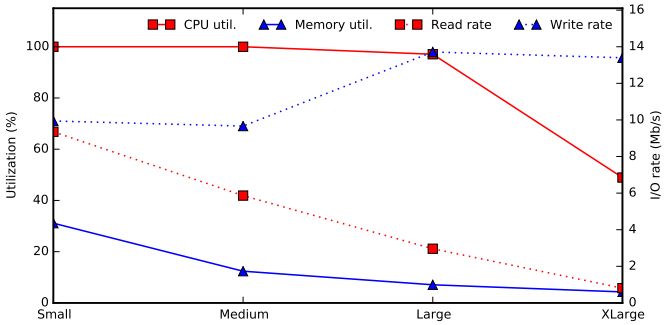
In our next experiment we investigate the impact of VM capacity on workflow performance. As mentioned in Section 4, VM capacity is represented by different VM types in ExoGENI, increasing as the type ranges from small to extra-large. The genomic workflows are executed on the *standalone* infrastructure, and only run on large and extra-large VMs because it requires at least 3GB RAM for data caching. The Montage workflows are run in the *multinode* infrastructure with 5 VMs interconnected with 100Mb/s network link. The experiments are performed on the WSU site at Detroit, MI and assigned to one workflow instance. As we observe in Figure 9, the walltime for both workflows decreases as VM capacity increases. This follows intuition since both workflows have a significant CPU and memory demand thus benefit from larger allocation of computing resources. Interestingly, as shown in Figure 9,



(a) Walltime as a function of VM type.



(b) System-centric performance of genomic workflow



(c) System-centric performance of Montage workflow

Figure 9: Performance as a function of VM capacities

the walltime of the computing-intensive genomic workflow is only improved by 3% when VM type changes from large to extra-large. By analyzing the system resource usage shown in Figure 9b, we find that the disk read rate had a negligible increase (write rate) as other metrics decreased significantly when the VM capacity is larger. The observation suggests the workflow performance is restrained by the I/O bottleneck at this stage. It is also noticed that the walltime of Montage workflow drops drastically by 59% when the VM type varies from medium to large but declines slightly by 14% with larger VMs. We also observe that CPU utilization even slumps by 49% when extra-large VM is used. The observation indicates that the large VM is optimal in cost performance for the Montage workflow and investment in more powerful nodes would bring marginal performance gain.

We now consider performance as a function of the number of VMs in the virtual infrastructure. We perform experiments on the *multinode* infrastructure with large VMs on the UCD site. The VMs are interconnected with 100Mb/s network link with its number ranging from 1 to 5. Because a genomic workflow consists of serial jobs and can only claim one slave node at a time, we assign an ensemble of 5 workflow instances to every experiment in order to utilize all the VMs. Each experiment with Montage workflow is assigned with one single workflow instance as its jobs can be

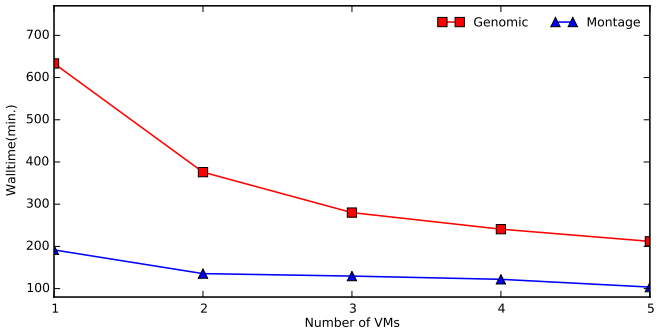


Figure 10: Walltime with varying number of VMs

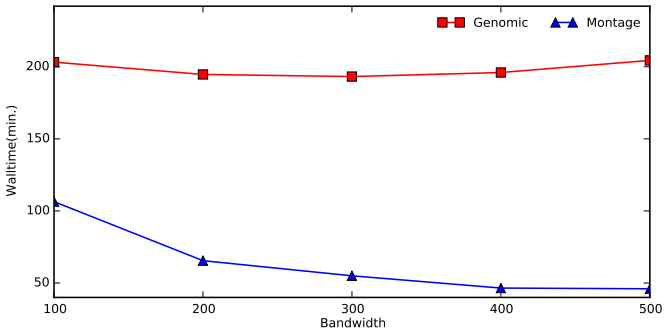


Figure 11: Walltime with varying network capacity

distributed over the VMs due to its parallelism. As it can be seen in Figure 10, the walltime is negatively correlated with the number of VMs for both workflows. The walltime is reduced by 66% and 45% for the genomic and Montage workflow respectively as the number of VMs is incremented from 1 to 5. Therefore, it is more cost-effective to scale up computing resources for the genomic workflow than the Montage workflow in improving workflow performance.

Finally, we investigate the impact of network capacity on workflow performance. We added a VM within the *slice* for data staging. We then varied the bandwidth of the network links connecting the staging site to the rest of the *slice* between 100Mb/s and 500 Mb/s. We ran the genomic workflow on one large VM and the Montage workflow on 5 large VMs *multinode* infrastructure. Figure 11 depicts the results. The average wall time of the genomic workflow decreases by 1% as the network capacity increases, while the Montage workflow improves in 56%. This result suggests that the Montage workflow benefits more from larger network allocation as compared to the genomic workflow. This is expected since the genomic workflow has a limited amount of intermediate data. These results demonstrate that *WRAP* provides scientists with *effective* control over the performance of their workflows via simple programmatic mechanisms, i.e., modifying resource capacity in the WVA file.

6. CONCLUSION

We explore the space of workflow repeatability on virtualized environments. We argue that the concept of virtual appliance is crucial to enable high-fidelity workflow re-execution in practice. We incarnate this principle into a novel, generic and extensible architecture that builds around a novel workflow appliance (WVA). We realize this architecture into *WRAP*;, a real system which builds on a IaaS and a workflow management system widely adopted in the scientific community. We demonstrate via experimentation on a production environment that *WRAP* is able to replay workflows with high fidelity under various conditions including higher level of resource heterogeneity and different times. Furthermore, *WRAP* bestows effective support for workflow predictability with performance information.

7. REFERENCES

[1] Montage. http://montage.ipac.caltech.edu/docs/grid.html.

[2] Network descriptive language. http://en.wikipedia.org/wiki/Network_Description_Language.

[3] UC Davis, UC Santa Barbara, and UC San Diego. https://kepler-project.org/.

[4] S. Bechhofer, J. Ainsworth, J. Bhagat, I. Buchan, P. Couch, D. Cruickshank, D.D. Roure, M. Delderfield, I. Dunlop, M. Gamble, C. Goble, D. Michaelides, P. Missier, S. Owen, D. Newman, and S. Sufi. Why Linked Data Is Not Enough for Scientists. In *e-Science (e-Science), 2010 IEEE Sixth International Conference on*, pages 300–307, 2010.

[5] K. Belhajjame, C. Goble, S. Soiland-Reyes, and D. De Roure. Fostering Scientific Workflow Preservation through Discovery of Substitute Services. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 97–104, 2011.

[6] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of Scientific Workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10. IEEE, 2008.

[7] Jeff Chase, Laura Grit, David Irwin, Varun Marupadi, Piyush Shivam, and Aydan Yumerefendi. Beyond Virtual Data Centers: Toward An Open Resource Control Architecture. In *in Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library), ACM*, 2007.

[8] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming*, 13(3):219–237, 2005.

[9] ExoGENI. http://www.exogeni.net/.

[10] Juliana Freire, Philippe Bonnet, and Dennis Shasha. Computational Reproducibility: State-of-the-art, Challenges, and Database Research Opportunities. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 593–596, New York, NY, USA, 2012. ACM.

[11] Ian P. Gent. The Recomputation Manifesto. http://arxiv.org/abs/1304.3674, April 2013.

[12] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the Challenges of Scientific Workflows. *Computer*, 40(12):24–32, 2007.

[13] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[14] Anirban Mandal, Paul Ruth, Ilya Baldin, Yufeng Xin, Claris Castillo, Gideon Juve, Mats Rynge, Ewa Deelman, and Jeff Chase. Tr-15-01: Adapting Scientific Workflows on Networked Clouds Using Proactive Introspection. Technical Report TR-15-01, Renaissance Computing Institute (RENCI), 2015.

[15] MongoDB. https://www.mongodb.org/.

[16] Ahalt S. Berg J. Coyle J. Evans J. Fecho K. Gillis D. Schmitt C. Young D. Owen, P. and K. Wilhelmsen. Technologies for Genomic Medicine: The GMW, A Genetic Medical Workflow Engine. 2014.

[17] RabbitMQ. http://www.rabbitmq.com/.

[18] David De Roure, Carole Goble, and Robert Stevens. The Design and Realisation of the Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems*, 25(5):561 – 567, 2009.

[19] Idafen Santana-Perez, Rafael Ferreira da Silva, Mats Rynge, Ewa Deelman, MarÃŋaS. PÃĺrez-HernÃąndez, and Oscar Corcho. A Semantic-Based Approach to Attain Reproducibility of Computational Environments in Scientific Workflows: A Case Study. In *Euro-Par 2014: Parallel Processing Workshops*, volume 8805 of *Lecture Notes in Computer Science*, pages 452–463. Springer International Publishing, 2014.

[20] Constantine Sapuntzakis and Monica S. Lam. Virtual Appliances in the Collective: A Road to Hassle-Free Computing. In *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*, HOTOS'03, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[21] Victoria Stodden, Freidrich Leisch, and Roger D. Peng. *Implementing Reproducible Research*, chapter 10: Reproducibility, Virtual Appliances, and Cloud Computing, pages 282–295. CRC Press, 2014.

[22] Indiana University. FutureGrid. https://portal.futuregrid.org/.

[23] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 2013.

[24] Jun Zhao, J.M. Gomez-Perez, K. Belhajjame, G. Klyne, E. Garcia-Cuesta, A. Garrido, K. Hettne, M. Roos, D. De Roure, and C. Goble. Why Workflows Break? Understanding and Combating Decay in Taverna Workflows. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–9, 2012.