Toward Prioritization of Data Flows for Scientific Workflows Using Virtual Software Defined Exchanges

Anirban Mandal, Paul Ruth, Ilya Baldin
RENCI - UNC Chapel Hill
{anirban, pruth, ibaldin}@renci.org

Rafael Ferreira da Silva, Ewa Deelman
USC Information Sciences Institute
{rafsilva, deelman}@isi.edu

***Abstract*—Recent advances in cloud systems, on-demand circuits and software-defined networking have created new opportunities to enable complex, data-intensive scientific applications to run on dynamic networked cloud infrastructures. In this work, we present an end-to-end framework for autonomic adaptation for scientific workflows on networked cloud systems, which leverages novel network provisioning technologies. We present an application-independent controller framework called Mobius++ that includes dynamic network adaptation capabilities using Software-Defined Networking (SDN) mechanisms, which enables workflow management systems to address competing priorities of workflow operations, data movements in particular. We use a representative, data-intensive bioinformatics workflow as a driving use case to showcase the above capabilities. Experimental results show that the Mobius++ framework, in conjunction with a novel virtual Software Defined Exchange (SDX) platform, is able to dynamically prioritize bandwidths between different endpoints, on-demand, and being driven by priority directives from a workflow management system. We show that data transfer jobs from two workflows with different priorities are accurately arbitrated as the relative priorities change.**

***Index Terms*—scientific workflows; networked clouds; data flow prioritization; software defined exchange**

I. Introduction

Advanced hardware and software infrastructures have made it possible to deploy and run the critical data processing and analysis applications that arise in many fields of science and engineering. Examples of such applications include those developed by bioinformaticians to extract information out of enormous sequence datasets, those developed by physicists to analyze data produced by high-end instruments such as particle accelerators and telescopes, or those developed by data scientists to analyze Web and social networking data. Workflows have emerged as a flexible representation to declaratively express such complex applications with data and control dependencies, and have become mainstream in domains such as astronomy, physics, climate science, earthquake science, biology, and others [1], [2].

Large-scale computations are often composed of several interrelated workflows grouped into collections consisting of workflows that have a similar structure, but may differ in their input data, number of tasks, and individual task sizes. In some cases, workflows within such a collection may have different priorities, for example, to access/transfer data from external repository or datastore. It might also be the case where different workflows, say a hurricane simulation workflow and a bioinformatics workflow, share a common computational and network infrastructure, and their computations and data flows might need to be arbitrated depending on external factors like an impending hurricane. Therefore, it is crucial that the computational and network infrastructures provide mechanisms to address competing priorities of workflow operations such as data acquisition from external storage resources, and to provide support for dynamic computational needs as the workflows proceed through execution stages. It is also essential to tailor and adapt the infrastructure based on the requirements of the workflows.

In recent years, cloud Infrastructure-as-a-Service (IaaS) systems [3], [4] have been designed to offer virtualized infrastructure as a unified hosting substrate for diverse applications. Similarly, network substrates increasingly offer control interfaces for dynamic virtualization (e.g., circuits and software-defined networking). These advances have created an unprecedented opportunity to enable data-intensive scientific applications on elastic networked cloud infrastructure. We refer to this model as Networked Infrastructure-as-a-Service (NIaaS). Such infrastructures link distributed resources into connected arrangements called *slices*, which are mutually isolated pieces of networked virtual infrastructure, carved out from multiple cloud and network transit providers, and is built to order for guest applications like scientific workflows. One such exemplar NIaaS system used in this work is ExoGENI [5] (Section IV-A).

ExoGENI provides many of the low-level mechanisms for on-demand infrastructure creation and adaptation of network and compute resources. But, the resource and request representations used in NIaaS systems, for e.g. RSpec [6] and NDL-OWL [7], are designed for the purpose of infrastructure management — resource provisioning, allocation, etc. — but are not suitable for direct use by higher-level applications. Hence, in our previous work [8], we developed an application-independent controller framework called Mobius that can translate high-level resource requirements to low-level NIaaS operations to instantiate appropriate resource envelopes for workflow execution. While Mobius had support for creation of simple connected slices (e.g., HTCondor pools with data

links) and for providing dynamic computational elasticity during workflow execution through computational infrastructure adaptation, it lacked support for new kinds of mechanisms developed in ExoGENI and needed by workflows — network adaptation (bandwidth shaping, flow control), dynamic addition of network links, and new Software-Defined Networking (SDN) capabilities like Software-Defined Exchanges [9].

In this work, which is done in the context of the DoE Panorama project [10], [11], we present an end-to-end framework for autonomic adaptation of complex scientific workflows on networked cloud systems driven by the requirements from workflow management systems like Pegasus [12], [13]. Our particular emphasis is on novel network provisioning mechanisms and how we can extend these capabilities up to scientific workflow management systems. We present a new version of Mobius, called Mobius++, which includes new dynamic network adaptation capabilities like virtual Software Defined Exchanges (SDX) that enables workflow management systems to address competing priorities of workflow operations, both data movement and computational. To demonstrate our new capabilities with a real use case, we show how data flows resulting from workflow data transfers can be arbitrated transparently based on workflow priorities for a data-intensive bioinformatics workflow.

This paper is organized as follows. Section II presents the overall system architecture including the Mobius++ framework with support for virtual SDX and a network adaptation use case for scientific workflows. An experimental evaluation of the system with a real data-intensive workflow instantiated on ExoGENI is presented in Section III. Section IV provides background on ExoGENI, Pegasus workflow management system, and the Panorama project. Section V presents related work, and Section VI concludes the paper and discusses future work.

II. System Description

In our previous work, we built an application-independent controller framework called Mobius [8] that works in concert with an application-aware controller such as a workflow management system. It takes a description of a desired high-level resource configuration for the slice, and issues requests to the underlying Networked Infrastructure-as-a-Service (NIaaS) system (ExoGENI) to (re)configure and adapt the slice to implement the specified changes. Mobius consumes high-level application specific requests, automatically transforms them to appropriate low-level infrastructure provisioning requests, runs policies to adapt the infrastructure, and adjusts resource allocations based on the demands from the application. The purpose of Mobius is to facilitate ease of use of NIaaS systems for workflow systems. One of the key design principles for Mobius was the separation of concerns between the application-aware controller, which is application aware but NIaaS agnostic; and the application-independent controller, which is NIaaS aware but application agnostic.

A. Mobius++

Building on the same design principle, we built an extended version of Mobius, referred to as Mobius++, to address new capabilities for resource provisioning driven by new types of requirements from the workflow management system. While Mobius includes capabilities for providing computational elasticity and addition of stitchports (network links to external data sources) to workflows, it does not provide any capabilities for adaptations of network performance in response to changing needs from the workflow management system. In order to extend these capabilities to workflow management systems layer, we developed new underlying NIaaS mechanisms for performing different kinds of network adaptation (e.g., bandwidth shaping, flow control, etc.) using novel network provisioning techniques like Software Defined Exchanges (SDX). These additional capabilities also required rethinking of how ExoGENI slices can be programmatically controlled to create new kinds of topologies, as well as how to support dynamic adaptation of network performance.

Hence, Mobius++ includes the following additional elements: (1) an enhanced programmatic toolkit for interacting with ExoGENI slices, which we refer to as the Ahab Library (Section II-B); (2) addition of support for virtual SDX and its representative use in the context of workflows (Section II-C); and (3) an end-to-end architecture that captures the interactions among these new capabilities (Section II-D). In this section, we also describe how novel Mobius++ capabilities enable network adaptations for prioritized flow requests originated from the workflow management system (Section II-E).

B. Abab Library

With the advance of ExoGENI capabilities to include control facilities (e.g., Mobius++) to automatically modify provisioned resources, there is an urge for a programmatic toolkit for interacting with ExoGENI slices. *Ahab* is a graph-based Java library designed to allow applications to control and modify ExoGENI slices. The Ahab library is composed of a collection of libraries organized as follows:

1) ***libtransport***: A library that provides an abstraction for interacting with ExoGENI through its XMLRPC interface. This library requires users to provide their GENI [14] credentials in order to create a proxy object associated with a particular ExoGENI controller. The proxy can then be used to create, terminate, modify, or query any *slice* (or set of slices) owned by the tenant on a determined controller.
2) ***libndl***: A library that provides a graph-based abstraction for interacting with ExoGENI *slices*. It primary handles the conversion of the topology graph into NDL-OWL requests, which are then consumed by ExoGENI. *ComputeNodes* and *networks* can be added to a *slice* where they become nodes in the graph (more advanced node types include *stitchports* and *storage*). Nodes and networks can be connected by *stitching* nodes and networks together forming edges in the graph. Finished topologies

```
1  ITransportProxyFactory ifac = new XMLRPCProxyFactory();
2  TransportContext context = new PEMTransportContext("",pem,pem);
3  ISliceTransportAPIv1 sliceProxy = ifac.getSliceProxy(context,controllerURL);
4  Slice s = Slice.loadManifestFile(sliceProxy,sliceName);
5  ComputeNode newnode = s.addComputeNode("ComputeNode0");
6  newnode.setImage(imageURL,imageHash,imageName);
7  newnode.setNodeType("XO Large");
8  newnode.setDomain("RENCI (Chapel Hill, NC USA) XO Rack");
9  BroadcastNetwork net = (BroadcastNetwork)s.getResourceByName(netName);
10 Interface int1 = net.stitch(node);
11 ((InterfaceNode2Net)int1).setIpAddress(ip);
12 s.commit()
```

Fig. 1. Ahab code example (`pem`: user's pem file; `controllerURL`: URL to the ExoGENI Controller).

are *committed*, which pushes any changes to ExoGENI via the libtransport proxy.

3) ***libndl.extras***: *Extras* is a set of extensions to the libndl library that provides higher-level abstractions that the user can invoke in a native way. These extensions are typically sets of ExoGENI resources that act as a more sophisticated larger resource. For example, *extras* includes a class called *PriorityNetwork* that is used like a standard *BroadcastNetwork* but allows the user to set priority paths for traffic going through the network. When a user creates a *PriorityNetwork*, several nodes and networks are created utilizing OpenFlow — a virtual Software Defined Exchange, which will enforce the priority policies. The key idea is that the user gets this functionality without needing to know the internals of the *PriorityNetwork*. More details about the *PriorityNetwork and virtual SDX* will be discussed in the upcoming section.

Figure 1 shows a code snippet of how to use the basic features of libtransport and libndl. Lines 1-3 show how the user can create a *SliceProxy* that is used to communicate with the ExoGENI controller. Line 4 show how to query the ExoGENI controller for the slice with a specific name. The controller returns the NDL-OWL manifest of the slice which is converted into a graph that can be manipulated by the user. Line 5 shows how a user can add a new node to the graph, and lines 6-8 show how to set the properties of the new compute node. Line 9 shows how to get the network object from the slice graph. At this point, the user has a reference to both the new compute node and the existing network. Line 10 depicts the process for stitching the new node to the network by creating an interface between them. Line 11 sets the IP address of the interface. Last, line 12 commits the changes to the slice.

C. Virtual Software Defined Exchange (vSDX)

Traditionally, Internet Exchange Points (IXPs) are physical locations that provide services that enable independent network domains to exchange network traffic. Typically, IXPs work at layer-3 and route traffic between domains using policies determined by protocols such as BGP (Border Gateway Protocol). More recently, IXPs are incorporating software defined networking (SDN) to create software defined exchange points (SDXs) that allow for more fine-grained policies to be expressed by the IXP and the domains to which it connects [9].

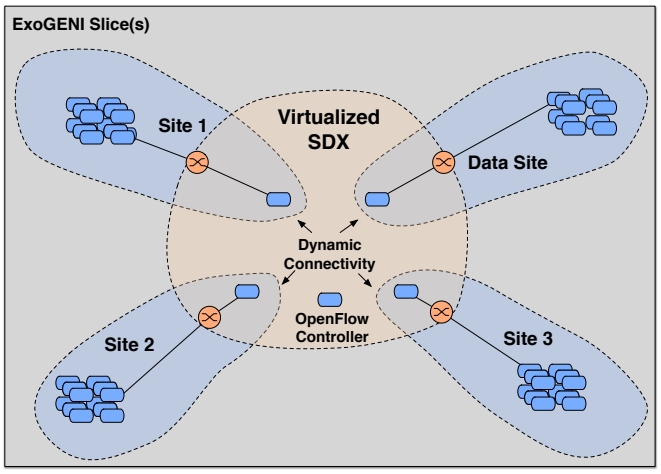

Fig. 2. Software Defined Exchanges (SDX): meeting point of networks to exchange traffic, securely and with QoS, using SDN protocols.

In the meantime, federated cyberinfrastructures, such as ExoGENI, have enabled many institutions to dynamically "stitch" their local network domains to complex topologies deployed across wide areas (regional, national, and international). This capability has made it possible to deploy an ExoGENI *slice* that accepts connections from many different institutional domains and uses SDN to forward network traffic between those domains. In effect, this network transit service acts like an SDX without a static physical location. In other words, the ExoGENI transit service slice (Figure 2) is a *virtual* SDX (vSDX) [15].

An ExoGENI vSDX is a dynamic slice rather than a static physical location. The vSDX can be created and managed by a controller application using the Ahab library described in Section II-B. Like a static SDX, a vSDX uses SDN within the exchange to enforce policies. However, the Ahab library can

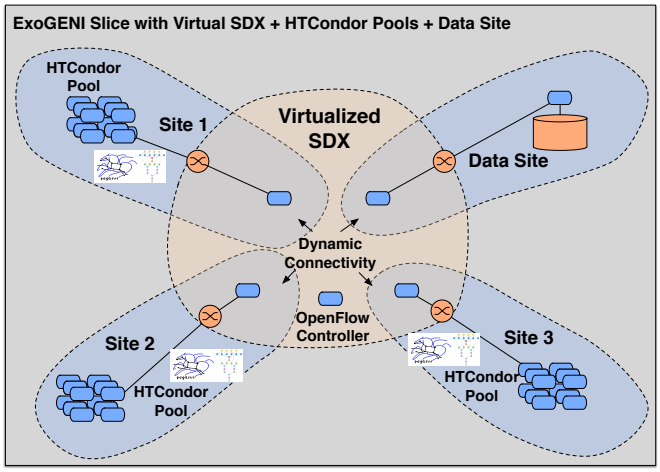

Fig. 3. vSDX use case with HTCondor pools and Pegasus.

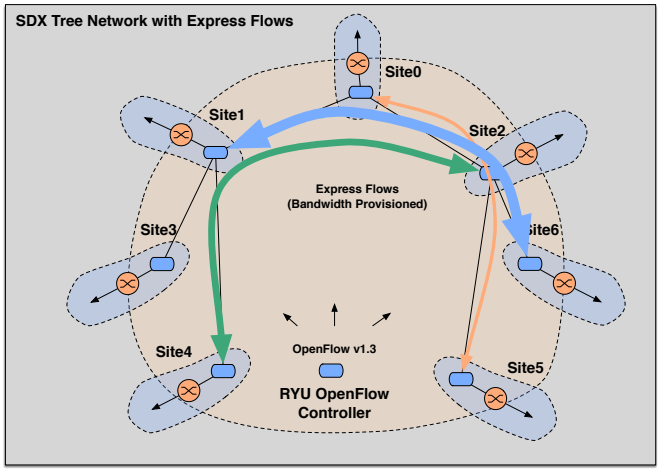

Fig. 4. Tree network with vSDX.

be used to automate modifications to the physical infrastructure used by the vSDX. These modifications can respond to changes in policy that require additional connectivity to new domains or even changes in the physical performance of a network path.

In the case discussed in this paper (Figure 3), a vSDX is used to connect a data repository and several HTCondor pools. This infrastructure is used to simultaneously support the execution of multiple Pegasus scientific workflows. The vSDX is based on Ahab's *PriorityNetwork* and is used by Mobius++ to enforce policy that prioritizes data transfer performance for each workflow based on dynamic application layer QoS requirements. Mobius++ communicates the data transfer priority requested by each workflow and sets the desired performance between the data repository and the HTCondor pool(s) supporting that workflow. Internally, the pools are connected to the *PriorityNetwork* which builds a tree of OpenFlow switches (Figure 4) that are able to forward traffic between any pair of sites. A Ryu [16] OpenFlow controller manages each of

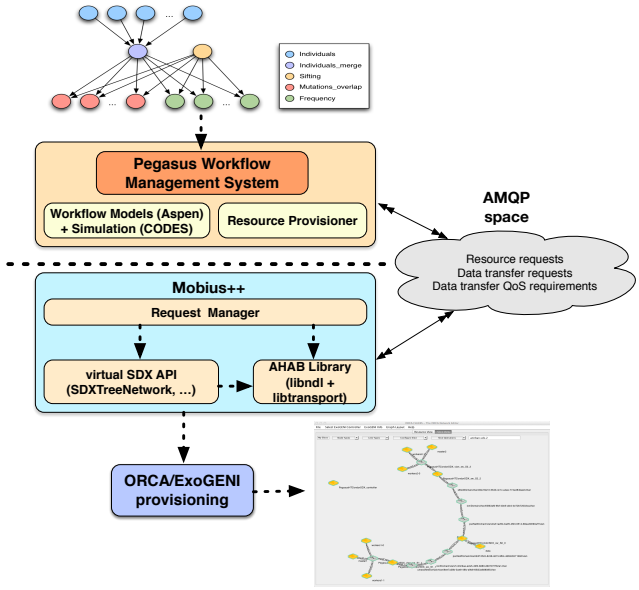

Fig. 5. Mobius++ end-to-end architecture.

the switches and installs queuing rules that create the desired bandwidth priorities between each site.

D. End-to-end Architecture

Figure 5 shows the end-to-end architecture consisting of the application-independent controller framework — Mobius++ that works in concert with the application-aware controller, and the Pegasus Workflow Management System. Pegasus takes as input the workflow description, and determines the workflow priorities using user directives, historical performance models, or by evaluating workflow performance models [10]. In addition to communicating compute requirements, as in Mobius, Pegasus can communicate data transfer requests and data flow QoS requirements to Mobius++ through an AMQP message space (RabbitMQ [17]). The high-level schema for the requirements included a new 'sdxcondor' initial request template and a new 'modifyNetwork' resource request in addition to other templates supported [8].

The *Request Manager* component in Mobius++ orchestrates the resource requirements communicated to the AMQP message space with the Ahab Library and the virtual SDX functionalities to send provisioning and de-provisioning requests to ExoGENI. It parses the high-level requests based on the request schema, and uses the APIs exposed by Ahab and virtual SDX to construct new and modify ExoGENI requests to instantiate new slices, modify existing topologies, and to modify properties in an existing slice. It is also responsible for sending the feedback to Pegasus through the AMQP message space when provisioning and modify actions are complete.

E. Adaptation with Prioritized Flows

The Mobius++ framework can be used by several high-level applications to provision and adapt infrastructure based on particular requirements. In this section, we present an example

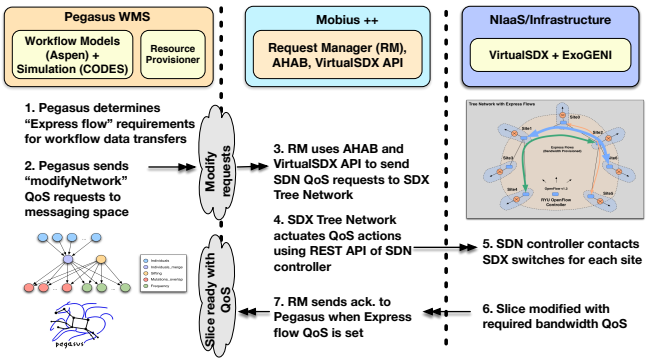

Fig. 6. Adaptation with Mobius++ – an example.

use case for Mobius++ framework for network adaptation for workflows, which is also used for evaluations (Section III). Figure 6 shows the timeline for the different steps involved in the adaptation process for the various entities: the Pegasus WMS, Mobius++, and the infrastructure provisioning system.

A slice with 'sdxcondor' template is created with two HTCondor pools, a data storage node, a virtual SDX, and an SDN controller node (This step is not shown in the timeline figure). While planning and executing the workflows on the HTCondor pools, Pegasus determines the flow requirements and priorities for workflow data transfers (step 1) and sends "modifyNetwork" QoS requests to the AMQP message space (step 2). The Request Manger component in Mobius++ uses the Ahab library and virtual SDX API to send SDN QoS requests (e.g., the flow priorities) to the SDX Tree network (step 3). The SDX tree network actuates the QoS actions by communicating with an SDN controller node using a REST API (step 4). The SDN controller then contacts the SDX switches in the virtual SDX to set OpenFlow rules, queues, and required attributes corresponding to the priorities requested (step 5). These actions result in modification of slice bandwidth across different sub-domains in the virtual SDX (step 6). Then the Request Manager sends an acknowledgement to Pegasus confirming that the flow requirements are satisfied. Pegasus continues with executing the steps of the workflow with the modified slice attributes and network properties (step 7).

III. Experimental Evaluation

In this section, we present an experimental evaluation of the data flow prioritization mechanism with Mobius++ powered by the virtual SDX system. We first describe a representative scientific workflow application in Section III-A. In Section III-B, we describe the experimental conditions and the infrastructure used to conduct the evaluations, and we present our experimental results and discussion in Section III-C.

A. Scientific Workflow Application

The 1000 genomes project provides a reference for human variation, having reconstructed the genomes of 2,504 individuals across 26 different populations [18]. The test case used

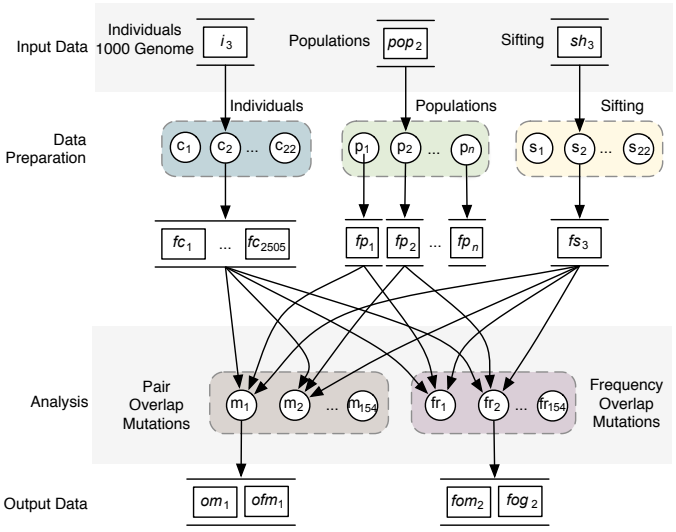

Fig. 7. Overview of the 1000 genome sequencing analysis workflow.

in this work identifies mutational overlaps using data from the 1000 genomes project in order to provide a null distribution for rigorous statistical evaluation of potential disease-related mutations. This test case (Figure 7) has been implemented as a Pegasus workflow [19], and is composed of five different tasks: (1) *individuals* – This task fetches and parses the Phase 3 data [18] from the 1000 genomes project per chromosome; (2) *populations* – The populations task fetches and parses five super populations (African, Mixed American, East Asian, European, and South Asian), and a set of all individuals; (3) *sifting* – This task computes the SIFT scores[1] of all of the SNPs (single nucleotide polymorphisms) variants, as computed by the Variant Effect Predictor; (4) *pair_overlap_mutations* – This task measures the overlap in mutations (SNPs) among pairs of individuals; and (5) *frequency_overlap_mutations* – This task calculates the frequency of overlapping mutations across subsamples of certain individuals.

The total data footprint for a typical run of the genomics workflow is about 4.4TB, and requires over 400GB of RAM (for the largest tasks – *individuals*). A detailed workflow characterization is available in [19]. In order to fit an instance of the workflow execution into our testbed (see description below), we have pruned the original datasets to process about 10% of the original data (about 11GB per individual dataset), and the processing of 2 populations. For this experiment, each workflow is composed of 22 *individuals* jobs, 2 *sifting* jobs, 14 *frequency_overlap_mutations* jobs, and 14 *pair_overlap_mutations* jobs. We will refer to this workflow as the 1000 genome workflow.

B. Experiment Conditions

The experiments used three racks from the ExoGENI testbed: a rack at the Lawrence Berkeley National Laboratory's

[1]SIFT is a sequence homology-based tool that Sorts Intolerant From Tolerant amino acid substitutions, and predicts whether an amino acid substitution in a protein will have a phenotypic effect.

Oakland Scientific Facility (OSF), Oakland, CA; a rack at the University of Massachusetts at Amherst (UMass), Amherst, MA; and a rack at the Pittsburgh Supercomputing Center (PSC), Pittsburgh, PA. Details about the hardware on these three racks can be found on the ExoGENI wiki [20]. Slices were provisioned from these racks by sending requests for virtual topologies consisting of (1) a set of virtual machines (VM) connected via a broadcast link with a specific bandwidth (250 Mb/s or 500 Mb/s) at each of the computing sites — OSF and UMass sites are used as computing sites (in each site we deploy an HTCondor pool with a master and several workers); (2) a VM hosting storage at the PSC site, which was used as the datastore node (emulates an external storage); and (3) a virtual SDX consisting of VMs acting as OpenFlow switches connecting the three sites in a tree network. A separate VM was provisioned at an ExoGENI rack in West Virgina (WVN), which performed the role of an OpenFlow controller. The overall bandwidth available on the virtual SDX connecting the three sites was configurable. Two values were used for the experiments – 500 Mb/s and 250 Mb/s.

WMS Configuration and Execution Environment. Since Pegasus uses HTCondor, the request to ExoGENI consisted of an HTCondor master VM and a specified number of HTCondor worker VMs connected by links with desired bandwidth for each rack hosting the HTCondor pools. The VM images had prerequisite software installed such as HTCondor, Pegasus, and the workflow application. The ExoGENI postboot script feature was leveraged to start various HTCondor daemons on VM startup so that the HTCondor environment is ready as soon as the slice setup is complete. An overview of the deployed environment consisting of two HTCondor pools and a data node, all connected through a virtual SDX, is shown in Figure 8. We have also leveraged the PRE script job capability of HTCondor/DAGMan to send "modifyNetwork" QoS requests (step 2, Figure 6), which holds the job execution until an authorization message is received from Mobius++ (i.e., the network is properly provisioned according to the priorities of the workflows currently running).

Determining Workflow Priorities. For each experiment run, we executed the 1000 genome workflow on each of the HTCondor pools using the Pegasus WMS. Two instances of the workflow were run simultaneously with different data flow *priority* values γ. Priority values are defined as follows:

$$\begin{aligned} \gamma(w_i) &= p \\ \gamma(w_j) &= 90 - p, \ i \neq j, \ \text{and} \\ & p \in \{10, 20, 30, 40, 45, 50, 60, 70, 80\}, \end{aligned} \tag{1}$$

where w_i and w_j are the two simultaneous workflow running instances, and p is the priority balancing factor between the workflows. For example, when one workflow requests a priority $\gamma = 20$, the corresponding second simultaneous workflow has a priority $\gamma = 70$. We reserved a default priority $\gamma_d = 10$ for other potential network traffic going through the virtual SDX, which is why the combined priority of the two

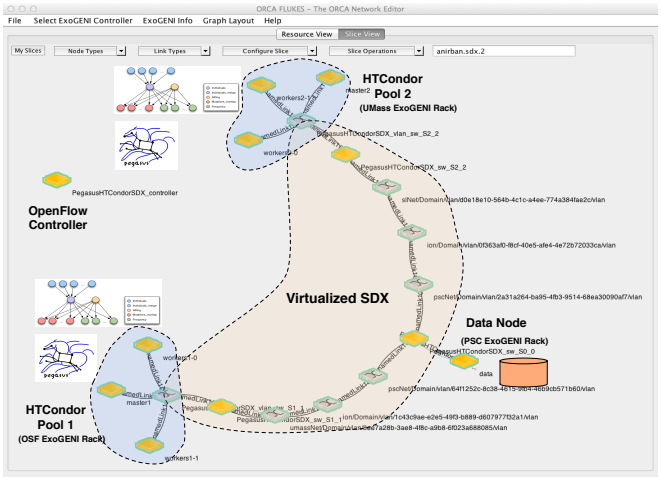

Fig. 8. HTCondor pools with Pegasus connected to a data store deployed with a virtual SDX on ExoGENI testbed racks.

workflows was set to 90. Tasks from both workflows were designed to fetch data from the common data store at the PSC rack. The average observed latency from the HTCondor master node in `HTCondor pool 1` (OSF rack) to the data node on the PSC rack was 71 ms. The average observed latency from the HTCondor master node in `HTCondor pool 2` (UMass rack) to the data node on the PSC rack was 20 ms. The *individuals* job is the most data intensive step in the workflow. Hence, the experimental results mainly focus on the average data transfer times for this step in addition to the overall workflow turnaround time (makespan).

Measured vs. Provisioned Bandwidth. Figure 9 shows the observed and provisioned bandwidths for one of the HTCondor pools (`HTCondor pool 1`) against the priorities requested by the workflow running on that pool (`Workflow1`). The top set of red lines correspond to the case where the overall virtual SDX bandwidth was set to 500 Mb/s, and the bottom set of blue lines correspond to the case for overall virtual SDX bandwidth of 250 Mb/s. The observed bandwidth was measured using iperf3 [21]. The provisioned bandwidth bw_p is calculated from the priority requested by `Workflow1`, the priority requested by workflow running on `HTCondor pool 2` (`Workflow2`), and the default priority γ_d reserved by the virtual SDX system:

$$bw_p(w_i) = \left(\frac{\gamma(w_i)}{(\gamma(w_i) + \gamma(w_j) + \gamma_d)} * bw \right), \tag{2}$$

where bw is the total vSDX bandwidth. For example, the provisioned bandwidth for a case where `Workflow1` priority $\gamma(w_1) = 30$ (with corresponding `Workflow2` priority $\gamma(w_2) = 60$, default virtual SDX priority $\gamma_d = 10$), and total bandwidth of 500 Mb/s, can be computed as follows:

$$bw_p(w_1) = \left(\frac{30}{(30 + 60 + 10)} * 500 \right) \text{ Mb/s}, \tag{3}$$

or 150 Mb/s. We observe that, for both the cases, the observed bandwidth is within 1–5% of the bandwidth provisioned by the

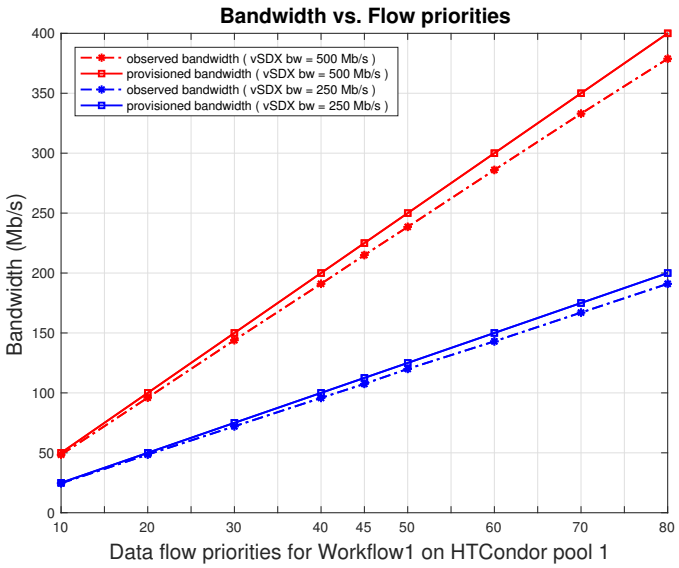

Fig. 9. Observed and provisioned bandwidths for different requested flow priorities by `Workflow1`. Observed bandwidth was measured using *iperf3*, while the provisioned bandwidth was computed with Equation 2.

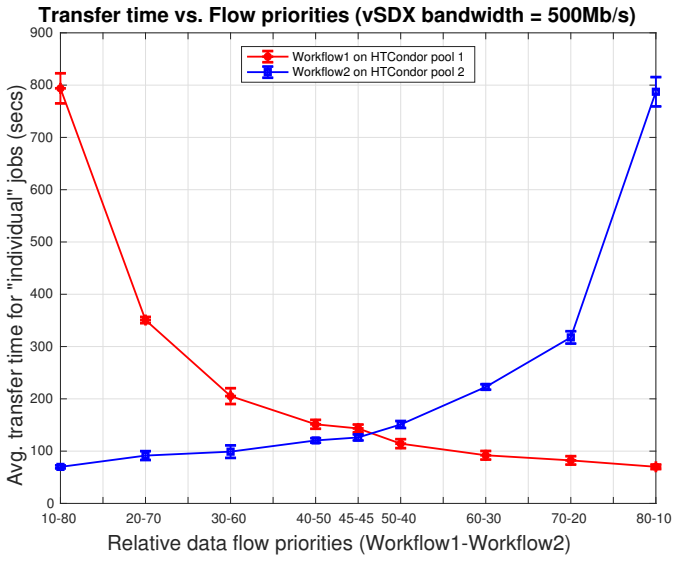

Fig. 10. Transfer time vs. relative flow priorities.

virtual SDX system. The results are similar when we plot the same with `Workflow2` on `HTCondor pool 2`. This shows that the priority requests are effectively satisfied by the virtual SDX system.

C. Experimental Results

We conducted sets of experiments using the experimental setup described above to study the prioritization capabilities of the virtual SDX system. We used the observed data transfer times of the data-intensive workflow steps, and the overall workflow execution time (i.e., workflow makespan) as important performance measures to evaluate the prioritization capabilities.

Workflow Data Transfer Times. Figure 10 shows the effect of relative flow priorities on the data transfer time as observed by data-intensive tasks for each of the two simultaneous workflows. The X-axis denotes the relative flow priorities of the two workflows for a particular run. The value "20-70" implies the case when Workflow1 was executed with requested priority of 20 and Workflow2 was executed with requested priority of 70. The virtual SDX system was responsible for arbitrating and carving out the overall available bandwidth of 500 Mb/s to the two workflows running on two HTCondor pools on two different sites. The Y-axis denotes the average data transfer time observed for the 20 *individuals* jobs in each of the two workflows. The two curves correspond to the two simultaneous workflows - Workflow1 and Workflow2. Each point is an average of 4 runs with corresponding error bars.

We observe that the virtual SDX system is able to prioritize the flows effectively. When the relative priorities of the two workflows differ by a large amount, we observe that the corresponding difference in transfer times is also large. For example, when Workflow1 has a priority of 10 and Workflow2 has a priority of 80, the average transfer time for Workflow1 is about 800 seconds while that of Workflow2 is about 70 seconds. The difference in average transfer times decreases as the difference between priorities of the two workflows decreases. This result shows that the virtual SDX system is able to arbitrate the available SDX bandwidth effectively between competing workflows.

Figure 11 shows the effect of relative flow priorities on the data transfer times for the *individuals* jobs for two different values of overall virtual SDX bandwidths - 500 Mb/s and 250 Mb/s. The X-axis and Y-axis denote relative flow priorities and average transfer times, as described in the previous result. The two sets of curves correspond to the different bandwidths — dotted curves representing the 500 Mb/s case and the solid curves representing the 250 Mb/s case. We observe similar behavior for the 250 Mb/s case as in the 500 Mb/s case described above, i.e. higher relative priority differences result in higher differences in transfer times. We also observe that the average transfer times also scale with the arbitrated bandwidth. Since the overall virtual SDX bandwidth available is half for the 250 Mb/s case, it is reflected in the average data transfer times being doubled when moving from 500 Mb/s to 250 Mb/s. This result shows that the virtual SDX system is able to arbitrate flows effectively in different bandwidth scenarios.

Workflow Makespan. Figure 12 shows the effect of relative flow priorities on the overall workflow execution times for the two workflows. The X-axis denotes the relative flow priorities of the two workflows for a particular run. The Y-axis denotes the workflow execution time (makespan) for each workflow. We observe that the execution time of a workflow decreases with increasing priority values with a corresponding increase of execution time for the other workflow with decreasing priority values. Since this workflow has a few data-intensive steps, the execution time is affected by the transfer times for all

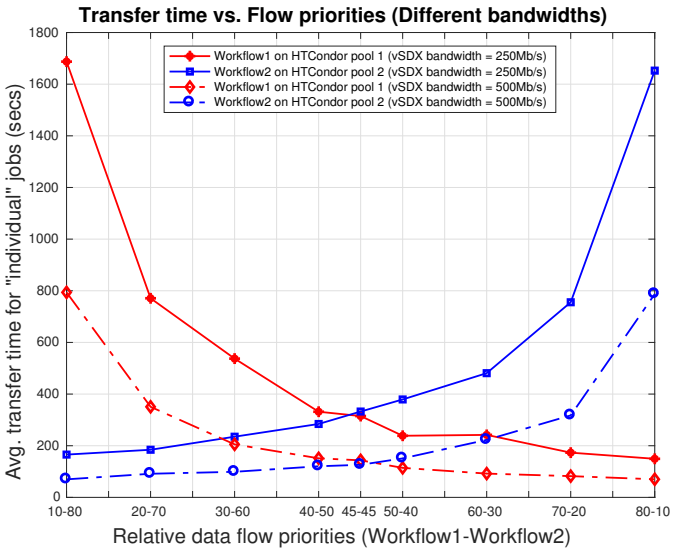

Fig. 11. Transfer time vs. relative flow priorities for different bandwidths.

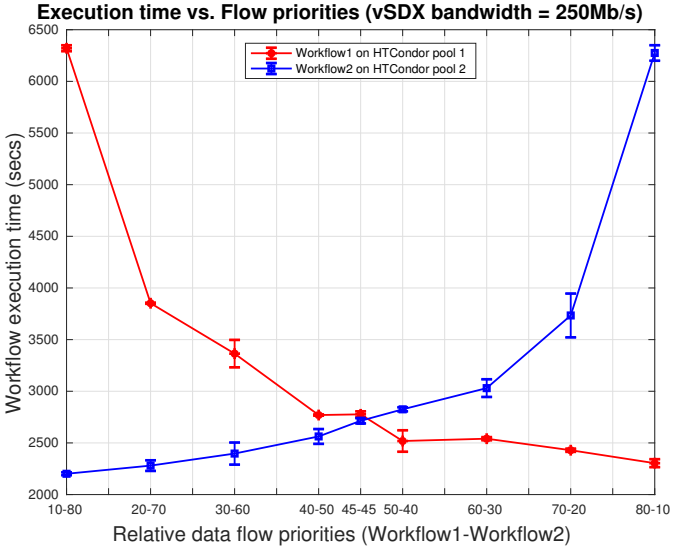

Fig. 12. Workflow makespan vs. relative flow priorities for different bandwidths.

the data-intensive steps in the workflow. This result shows that for data-intensive workflows, the virtual SDX system helps in overall turnaround time for higher priority workflows by arbitrating the transfers effectively.

IV. Background

A. ExoGENI

ExoGENI [5] is a network IaaS national testbed powered by the ORCA (Open Resource Control Architecture) [22] control software used for GENI. ORCA allows users to create mutually isolated "slices" of interconnected infrastructure from multiple independent providers (compute, network, and storage) and commodity infrastructure. To this end, ORCA takes advantage of existing virtualization mechanisms by integrating various resources together: Layer-2 global dynamic-

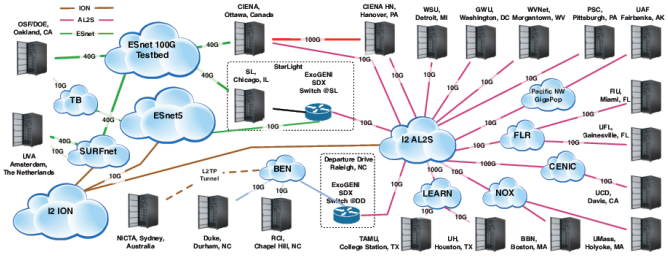

Fig. 13. ExoGENI testbed.

circuit networks like Internet2/ION, AL2S/OESS, and ESnet, software defined networks using OpenFlow, and private clouds like OpenStack, and xCAT. As shown in Figure 13, ExoGENI links racks at 20 sites on campuses and labs across the country through regional and national transit networks (Internet2, ESnet, etc.). Each rack is a small networked cloud, consisting of servers virtualizable via OpenStack and xCAT, sliverable storage and a high-performance (10G/40G) dataplane switch capable of both VLAN-based and OpenFlow operation. ExoGENI is also complemented by racks contributed by individual campuses (e.g., in Sydney, Australia, Amsterdam, The Netherlands, and others). ExoGENI racks are integrated into campuses and offer connectivity to select campus resources where the racks are deployed. This ties data transport capability with local computational resources in the rack, allowing to move some computations close to the data on campus or ingest it into a slice using a dynamic on-ramp-on-demand Layer2/SDN connection.

B. Pegasus WMS

The Pegasus workflow management system (WMS) [12], [13] provides the necessary abstractions for scientists to create workflows, allow for transparent execution of these workflows on a range of compute platforms including campus clusters, clouds, and grids. Since its inception 15 years ago, Pegasus has become an integral part of the production scientific computing landscape in several scientific communities [13]. In this work, we have used Pegasus as a representative workflow management system leveraging NIaaS platforms to plan and execute workflows.

In Pegasus, workflows are described abstractly as directed acyclic graphs (DAGs), where nodes represent individual computational tasks and the edges represent data and control dependencies between tasks, without any information regarding physical resources or physical locations of data and executables. During execution, Pegasus translates the abstract workflow into an executable workflow, determining the executables, data, and computational resources required for the execution. Workflow execution with Pegasus includes data management, monitoring, and failure handling. Individual workflow tasks are managed by a task scheduler (HTCondor [23]), which supervises their execution on local and remote resources.

C. Panorama Project

In the Panorama project [10], [11], we examine challenges present in the execution of large-scale workflow applications on world-class leadership machines, and we seek to automate the process of performance modeling, resource provisioning, workflow execution, and anomaly detection, which could lead to resource selection at runtime. This approach integrates extreme-scale systems, testbed experimentation, structured analytical modeling and parallel systems simulation with the Pegasus WMS. The Panorama project pursues a multi-prong approach: it collects runtime information about the execution of workflows in distributed and heterogeneous environments, it instruments and collects information from the infrastructure, it uses analytical and simulation-based models of the workflow execution on diverse resources, and it compares the models to the observed behavior during workflow execution and based on that comparison it detects anomalies in the execution and adapts the workflow and the infrastructure.

V. Related Work

In the past years, cloud computing has become a major topic of discussion in the scientific computing community. More specifically, many recent studies have focused on investigating the effectiveness and applicability of IaaS cloud platforms for executing scientific workflows [24]–[26]. In [25], a survey of public cloud elasticity for scientific applications shows that most distributed large-scale applications strongly depend on automated infrastructure optimizations (i.e., the application is unaware of the platform capabilities) to maximize resource usage, while reducing costs and meeting deadlines. Although scientific workflows provides a significant step toward automation of computation and data management for scientific applications, there are still steep challenges that need to be addressed. Unsurprisingly, resource management for distributed applications has been the subject of enormous amounts of efforts [27]–[31], both from practitioners and from researchers, and there is arguably a clear disconnect between theory and practice [32]. Theoretical results are obtained based on strong and simplifying assumptions, so that resource management problems are rendered tractable.

Workflow management systems have also focused on the execution and management of distributed applications on clouds (both academic and commercial platforms) [33]–[36]. However, most of the strategies used by such systems target the deployment of virtual machines in the cloud platform (with limited support to dynamic elasticity), while none or minimal support to infrastructure optimization is enabled. In particular, data placement/movement and network configuration/provisioning decisions are crucial to achieve high performance, given the increasing size of datasets processed by scientific workflows, many of which now fall in the loosely defined *big data* category of applications [24], [37]. A recent survey [38] on dynamic workflows and user steering techniques underlines the need for monitoring awareness and data analysis to enable adaptation. An outcome of the survey is that most workflow management systems do not provide a complete set of tools and mechanisms to enable task and data configuration refinement (e.g., add/delete/replace tasks, change platform conditions, etc.). In a previous work [8], we presented a scheduling technique that predicts dynamic resource needs using a workflow introspection technique to be used within the Mobius system to actuate resource adaptation in response to dynamic workflow needs. This work is a first step towards dynamic adaptation (in particular elasticity), but it did not account for data flows and network adaptation.

VI. Conclusions and Future Work

We presented an application-independent controller framework called Mobius++ that facilitates the use of advanced network provisioning technologies for scientific workflows. Mobius++ consists of an enhanced programmatic toolkit for creating and modifying ExoGENI slices (the Ahab library), and a virtual SDX abstraction created using Ahab, which enables creation of *PriorityNetwork* by leveraging SDN technologies. The *PriorityNetwork* construct is useful for arbitrating and prioritizing data flows from competing workflows, with priorities being communicated to Mobius++ by a workflow management system (Pegasus). We evaluated this end-to-end framework using a representative, data-intensive bioinformatics workflow on the ExoGENI testbed. Results show that our system is able to dynamically prioritize bandwidths between different endpoints for relevant workflow transfers, and the data transfer jobs from competing workflows with different priorities are accurately arbitrated as their relative priorities change.

In the future, we plan to develop accurate models to determine priorities automatically. We also plan to develop new mechanisms for adaptation like migration of workflows during execution in case of failure and poor performance, and the corresponding enhancements for Mobius++ to support this desirable feature from the perspective of a workflow management system. We also plan to develop new policies in Mobius++ that will integrate performance monitoring and performance data analysis to drive the adaptation process.

Acknowledgments

This work was funded by DOE contract number #DESC0012636, “Panorama—Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows”. We thank Rosa Filgueira, Ian M. Overton, and Erola Pairo-Castineira for their valuable help.

References

[1] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated, 2007.

[2] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, “Scientific workflows: Moving across paradigms,” *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 66, 2016.

[3] Amazon Elastic Compute Cloud (Amazon EC2), http://www.amazon.com/ec2.

[4] OpenStack Cloud Software, http://openstack.org.

[5] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills, “Exogeni: A multi-domain infrastructure-as-a-service testbed,” in *The GENI Book*. Springer, 2016, pp. 279–315.

[6] "RSpec," http://www.protogeni.net/ProtoGeni/wiki/RSpec.

[7] J. Ham, F. Dijkstra, P. Grosso, R. Pol, A. Toonk, and C. Laat, "A Distributed Topology Information System for Optical Networks Based on the Semantic Web," *Journal of Optical Switching and Networking*, vol. 5, no. 2-3, June 2008.

[8] A. Mandal, P. Ruth, I. Baldin, Y. Xin, C. Castillo, G. Juve, M. Rynge, E. Deelman, and J. Chase, "Adapting scientific workflows on networked clouds using proactive introspection," in *IEEE/ACM Utility and Cloud Computing (UCC)*, 2015.

[9] A. Gupta, M. Shahbaz, L. Vanbever, H. Kim, R. Clark, N. Feamster, J. Rexford, and S. Shenker, "Sdx: A software defined internet exchange," *ACM SIGCOMM*, 2014.

[10] E. Deelman, C. Carothers, A. Mandal, B. Tierney, J. S. Vetter, I. Baldin, C. Castillo, G. Juve, D. Król, V. Lynch, B. Mayer, J. Meredith, T. Proffen, P. Ruth, and R. Ferreira da Silva, "PANORAMA: An approach to performance modeling and diagnosis of extreme scale workflows," *International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 4–18, 2017.

[11] "PANORAMA: Predictive modeling and diagnostic monitoring of extreme science workflows," https://sites.google.com/site/panoramaofworkflows/.

[12] 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.

[13] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, no. 0, pp. 17–35, 2015.

[14] "The NSF GENI (Global Environment for Network Innovations) Project," http://www.geni.net/.

[15] Y. Yao, Q. Cao, J. Chase, P. Ruth, I. Baldin, Y. Xin, and A. Mandal, "Slice-based network transit service: Inter-domain l2 networking on exogeni," in *IEEE INFOCOM Workshop on Distributed Cloud Computing (DCC)*, 2017.

[16] "Ryu SDN framework," https://osrg.github.io/ryu/.

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

[18] T. . G. P. Consortium, "A global reference for human genetic variation," *Nature*, vol. 526, no. 7571, pp. 68–74, 2015.

[19] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, and M. Atkinson, "Using simple pid controllers to prevent and mitigate faults in scientific workflows," in *11th Workflows in Support of Large-Scale Science (WORKS'16)*, 2016, pp. 15–24.

[20] "ExoGENI Wiki," https://wiki.exogeni.net.

[21] "iperf3," http://software.es.net/iperf/.

[22] J. Chase, L.Grit, D.Irwin, V.Marupadi, P.Shivam, and A.Yumerefendi, "Beyond virtual data centers: Toward an open resource control architecture," in *Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library)*, May 2007.

[23] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience," *Concurrency and computation: practice and experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[24] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[25] G. Galante, L. C. E. De Bona, A. R. Mury, B. Schulze, and R. da Rosa Righi, "An analysis of public clouds elasticity in the execution of scientific applications: a survey," *Journal of Grid Computing*, vol. 14, no. 2, pp. 193–216, 2016.

[26] D. Poola, M. A. Salehi, K. Ramamohanarao, and R. Buyya, "A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments," *Software Architectures for Cloud and Big Data Book*, 2016.

[27] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[28] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015.

[29] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[30] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, 2017.

[31] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization," *Scientific Programming*, vol. 2015, p. 5, 2015.

[32] U. Schwiegelshohn, "How to design a job scheduling algorithm," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2014, pp. 147–167.

[33] E. Deelman, K. Vahi, M. Rynge, G. Juve, R. Mayani, and R. Ferreira da Silva, "Pegasus in the cloud: Science automation through workflow technologies," *IEEE Internet Computing*, vol. 20, no. 1, pp. 70–76, 2016.

[34] L. Yu and D. Thain, "Resource management for elastic cloud workflows," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 775–780.

[35] F. Oesterle, S. Ostermann, R. Prodan, and G. Mayr, "Experiences with distributed computing for meteorological applications: grid computing and cloud computing," *Geoscientific Model Development*, vol. 8, no. 7, pp. 2067–2078, 2015.

[36] M. Kasztelnik, E. Coto, M. Bubak, M. Malawski, P. Nowakowski, J. Arenas, A. Saglimbeni, D. Testi, and A. F. Frangi, "Support for taverna workflows in the vph-share cloud platform," *Computer Methods and Programs in Biomedicine*, 2017.

[37] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017.

[38] M. Mattoso, J. Dias, K. A. Ocana, E. Ogasawara, F. Costa, F. Horta, V. Silva, and D. de Oliveira, "Dynamic steering of hpc scientific workflows: A survey," *Future Generation Computer Systems*, vol. 46, pp. 100–113, 2015.