Adapting Scientific Workflows on Networked Clouds Using Proactive Introspection

A RENCI Technical Report

renci

www.renci.org

Adapting Scientific Workflows on Networked Clouds Using Proactive Introspection

Anirban Mandal, Paul Ruth, Ilya Baldin, Yufeng Xin, Claris Castillo
RENCI - UNC Chapel Hill
{anirban, pruth, ibaldin, yxin, claris}@renci.org

Gideon Juve, Mats Rynge, Ewa Deelman
ISI - USC
{juve, rynge, deelman}@isi.edu

Jeff Chase
Duke University
chase@cs.duke.edu

***Abstract*—Recent advances in cloud technologies and on-demand network circuits have created an unprecedented opportunity to enable complex data-intensive scientific applications to run on dynamic, networked cloud infrastructure. However, there is a lack of tools for supporting high-level applications like scientific workflows on dynamically provisioned, virtualized, networked IaaS (NIaaS) systems. In this paper, we propose an architectural framework consisting of application-aware and application-independent controllers that provision and adapt complex scientific workflows on NIaaS systems. The application-independent controller simplifies the use of NIaaS systems by higher-level applications by closing the gap between application abstractions and resource provisioning constructs. We also present our approach to predicting dynamic resource requirements for workflows using an application-aware controller that proactively evaluates alternative candidate resource allotments using workflow introspection. We show how these high-level resource requirements can be automatically transformed to low-level NIaaS operations to actuate infrastructure adaptation. The results of our evaluations show that we can make fairly accurate predictions, and the interplay of prediction and adaptation can balance performance and utilization for a representative data-intensive workflow.**

I. Introduction

The advent of pervasive virtualization has heralded an evolution from static arrangements of resources that persist over long periods of time to dynamically provisioned systems such as cloud infrastructure services. In recent years, cloud Infrastructure-as-a-Service (IaaS) systems have been designed to offer virtualized infrastructure as a unified hosting substrate for diverse applications [1], [2], and IaaS solutions have been deployed in public, private, and institutional clouds. Similarly, network substrates increasingly offer control interfaces for dynamic virtualization (e.g., circuits and software-defined networking). These advances, coupled with programmable edge technologies, 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).

Networked cloud infrastructures link distributed resources into connected arrangements, *slices*, targeted at solving a specific problem. This *slice* abstraction is central to providing mutually isolated pieces of networked virtual infrastructure, carved out from multiple cloud and network transit providers, and built to order for guest applications like scientific workflows (Figure 1). One such NIaaS system is ExoGENI [3], which uses the ORCA control framework [4] to create mutually isolated slices of interconnected virtual infrastructure from multiple clouds and network providers.

Data-driven workflows are becoming a centerpiece of modern computational and data-intensive science. One advantage of NIaaS is that it permits an application's slice to adapt elastically to balance performance and cost as the workflow executes. It opens the possibility to manage the performance of multiple workflows — and other kinds of applications—that share a common NIaaS infrastructure with different demands and priorities. To manage elastic slices, a *controller* drives adaptation of each slice as its resource demands and priorities change, while the NIaaS system arbitrates the requests of multiple controllers [5], [6], [7], [8]. Many systems have implemented elastic Web services using controllers [9] that react to changes in request arrival rate or resource utilization. Elastic provisioning for IaaS systems is an active research area [10]. There has been relatively less progress on adaptation for complex scientific workflows on cloud systems. This paper addresses that goal.

The resource and request abstractions used in NIaaS systems are designed for the purpose of infrastructure management — resource provisioning, allocation, etc.—but are not suitable for direct use by higher-level applications. Applications seldom require full visibility into these operations; instead they require intermediate abstractions crafted to their internal models so that their adaptations are guided by the specific workflow structure. In this paper, we make the case for a split controller framework combining an application-independent controller element with an application-aware element for *proactive introspection* into the workflow's structure and near-term resource demands. This combination closes the gap between application abstractions and resource provisioning constructs by enabling automatic mapping of complex applications to dynamic infrastructures.

We present a new approach to building application-aware controllers for computational workflows that are proactive and take advantage of application-specific knowledge. That knowledge is supplied as a declarative description of the workflow structure, together with performance models for the individual tasks. This approach allows the controller to be "clairvoyant" with respect to the workflow's resource needs as it determines how to adapt the slice. In particular, it can evaluate the projected performance of the workflow within alternative candidate resource envelopes as a basis for choices about how to adapt. Results show that this approach can be effective in balancing performance, resource utilization and

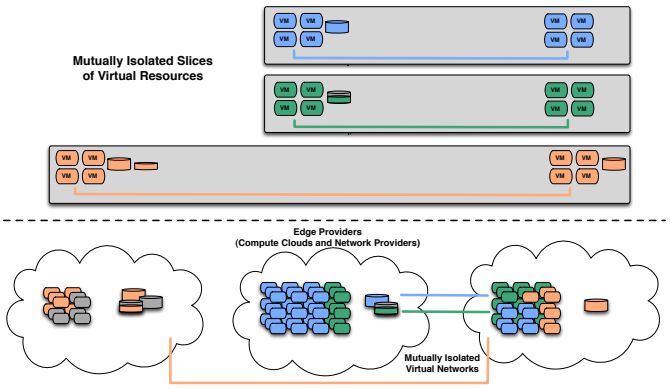

Fig. 1. Networked clouds

cost.

This paper describes an end-to-end system (Figure 2) that uses the Pegasus [11] workflow management system to launch workflows on HTCondor [12] environments deployed on the ExoGENI NIaaS testbed, and includes the application-aware and application-independent controllers for workflow introspection and infrastructure adaptation, respectively. The main contributions of the paper are

- We present our approach to predicting dynamic resource needs for workflows using an application-aware controller, **ShadowQ**, which proactively evaluates alternative candidate resource allotments using workflow introspection.
- We describe how these high-level resource requirements are transformed to low-level NIaaS operations to instantiate appropriate resource envelopes for workflows using an application-independent controller framework, **Möbius**.
- We present how these two controllers can work together, as an end-to-end system, to actuate infrastructure adaptation in response to application needs.
- We present an evaluation of (a) the accuracy of the predictions, and (b) the interplay of prediction and adaptation to balance performance and utilization.

II. Architecture

Two elements work together to support elastic workflow adaptation — (a) By leveraging the monitoring capabilities of Pegasus/HTCondor, the ShadowQ application-aware controller tracks the current state of the workflow, "introspects" on the workflow to project the remaining time to completion based on a workflow model, current workflow state and slice state, and predicts future resource requirements based on the introspections. (b) A general (application-independent) controller framework, Möbius, works in concert with ShadowQ: it takes a description of a desired high-level resource configuration for the slice, and issues requests to the underlying NIaaS system to reconfigure and adapt the slice to implement the specified changes.

A. *ShadowQ - Workflow Introspection*

The Pegasus workflow management system was extended by adding a new workflow introspection component called the ShadowQ. The ShadowQ monitors a running workflow, makes predictions about future events in the workflow by running discrete event simulations, and communicates resource requirements to the Request Manager, a component of Möbius.

In order to do this, the ShadowQ needs to keep track of the state of the workflow as it is running. It does this by reconstructing the state of the workflow using the Pegasus log files. The log files tell the ShadowQ which jobs are queued, which jobs are running, and which jobs have finished. They also indicate when jobs changed state. Using this information the ShadowQ can construct an accurate snapshot of the current state of the workflow at any time.

Starting with the current state of the workflow, the ShadowQ performs a set of discrete event simulations to predict future events in the workflow. The goal of these simulations is to predict two types of events: 1) the finish time of the workflow, and 2) the start times of the data staging jobs. These two event types are critical for provisioning the appropriate resources for the workflow. The finish time of the workflow indicates whether more resources are needed to complete the workflow by a given deadline or if fewer resources could be used, and the start time of the data staging jobs indicates when network links and data on-ramps need to be provisioned to accelerate the movement of data. Knowing the approximate time of these events in advance enables the provisioning system to ensure that the appropriate resources are allocated when they are needed. The ShadowQ runs for the duration of the workflow execution, and periodically re-evaluates its predictions by running additional simulations. Since simulations for relatively large workflows only take on the order of seconds to complete, these periodic simulations can occur every few minutes.

The simulations performed by the ShadowQ require estimates of the runtimes of the jobs in the workflow, and the size of the input and output files. These estimates can be obtained from user estimates, using performance models of the application, or based on historical data from previous runs of the workflow [13]. The ShadowQ can update these estimates based on new information as the workflow progresses.

The ShadowQ estimates not only what the finish time of the workflow will be given the currently available resources, but also the level of resources required to complete the workflow by a given deadline. It does this by performing a set of simulations that estimate the behavior of the workflow with varying levels of resources. Every simulation set begins with a simulation that estimates the finish time of the workflow with the currently available resources. This first simulation is also used to predict the starting time of staging jobs. If the workflow will not be able to meet the deadline given the currently available resources, or if the workflow will finish significantly sooner than the deadline, then the ShadowQ performs more simulations with different resource configurations to search for the minimum level of resources required to finish the workflow by the deadline. The resource configurations are set up as a binary search in order to minimize the number of simulations that need to be performed.

The starting times of the staging jobs are used to predict when network links and data on-ramps need to be provisioned. Based on the start time, duration, and the network resources

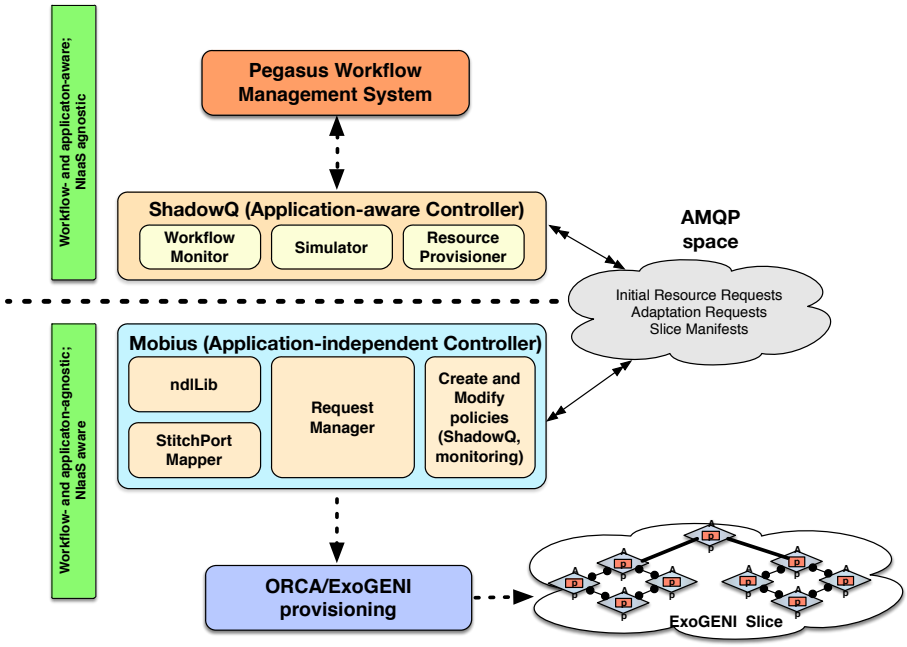

Fig. 2. Mapping applications to NIaaS.

used by each staging job, the ShadowQ can construct a schedule for provisioning them.

Resource requests are communicated from the ShadowQ to the Request Manager using a messaging system. The ShadowQ publishes messages to communicate the time when the workflow will finish given the currently available resources, the minimum number of resources required to finish the workflow by the deadline, and the amount of time until specific network links and data on-ramps will be required. The Request Manager responds with messages detailing the availability of compute and network resources.

B. *Möbius - Provisioning and Adaptation*

The application-independent controller, Möbius, consumes these high-level application specific requests (for eg. from ShadowQ), automatically transforms them to appropriate low-level infrastructure provisioning requests, runs policies to adapt infrastructure, and adjusts resource allocations based on the demands from the application. The purpose of Möbius is to facilitate ease of use of NIaaS systems for workflow systems, which enables mapping and adapting applications to these novel infrastructures, thereby closing the gap between application abstraction and resource provisioning constructs.

In our design of the end-to-end system (Figure 2), we made sure that there exists 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. We now describe the components of the application-independent controller.

1) ndlLib: The native request and resource representation used by ExoGENI is based on declarative representations using NDL-OWL. Although, this abstraction works well for topology embedding and provisioning resources at the NIaaS layer, these are not suitable for higher level applications. The applications are seldom interested in low-level topologies, and often have a simplistic view of the kind of resources they need. For example, the application layer often expresses resource requirements in terms of needing a “Condor pool”, “Hadoop cluster”, “distributed Condor pool with on-ramp to an external data-set”, “MPI cluster with low latency and high bandwidth”, etc. The goal of ndlLib, an essential component of Möbius shown in Figure 2, is to provide a library to map these application abstractions to NIaaS provisioning constructs to shield the users and applications from the details of the topology request.

The *ndlLib* library has a simple graph theoretic view, and a corresponding programmatic API for generating NDL-OWL requests. It allows applications to programmatically create a graph representation of desired compute, network, and storage resources. The graph can describe either a new slice of resources or a modification to and existing slice. The graph can then be saved as NDL-OWL or submitted directly to ExoGENI. With ndlLib, several common request templates are readily available for different application classes, and it becomes relatively easy for users to programmatically obtain NDL-OWL requests from a very high level application model.

2) Support for stitchport mappers and registries: Stitchports are ExoGENI entities that are used to access resources beyond the control of ExoGENI, e.g., data sets on campus resources. What lies beyond a stitchport is assumed to be IP-based infrastructure of a campus or a lab. One challenge of using stitchports is that attaching an ExoGENI slice with an existing local network requires coordination of network configuration. The ExoGENI slice should have conforming IP addresses that allow packets to be routable to and from the external resources and substrates. In order to aid the ExoGENI

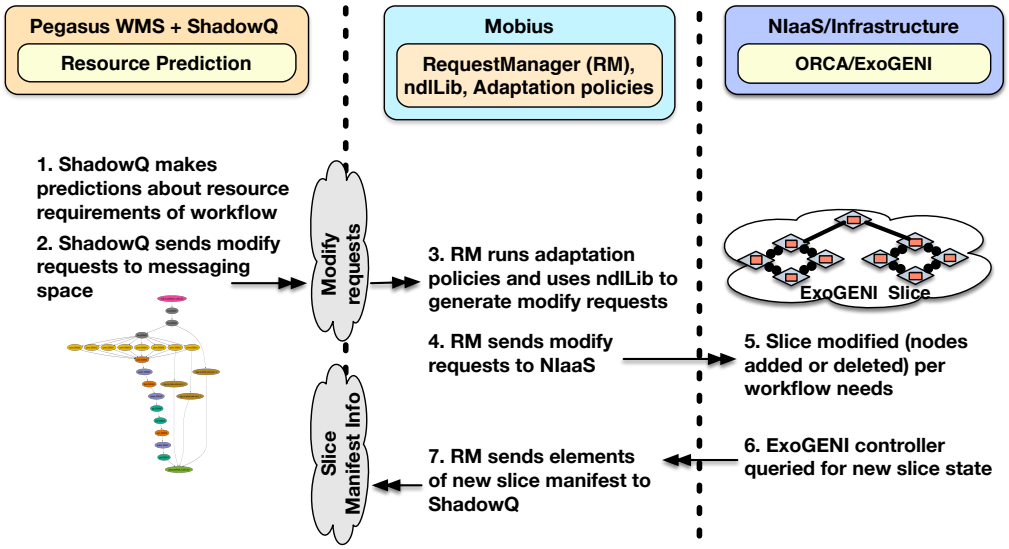

Fig. 3. Adaptation with ShadowQ and Möbius.

user in this task, ExoGENI provides a two-prong strategy – (a) an ExoGENI wide stitchport registry service, and (b) stitchport mapper services owned by the owners of the external resources. The stitchport registry contains the map from an identifier of a stitchport known to user to the URL of the stitchport mapper service. The stitchport mapper service allows the owner of a stitchport to advertise its availability and corresponding IP-level configuration to other ExoGENI users and applications.

The current implementation of the stitchport registry and mapper consists of a RESTful service with Apache CouchDB [14] backend for storing data. Each document (i.e., record in SQL-like jargon) in CouchDB consists of a stichport name and an URL configuration data service URL as key and value respectively. The URL data service serves the IP-level configuration information associated with the corresponding stitchport.

3) Request Manager: The Request Manager is an entity that orchestrates the above functionalities to construct and send provisioning and de-provisioning requests to ExoGENI. Based on requirements from the ShadowQ or high-level applications, optional monitoring data, the Request Manager invokes ndlLib and (optionally) the stitchport registry and mapper, constructs and annotates original and modify requests, and sends them to ExoGENI to instantiate or modify a slice.

Interaction with ShadowQ: The Request Manager interacts with the higher level applications (like the ShadowQ) to consume resource requirements and expectations. As shown in Figure 2, it is responsible for receiving messages from the ShadowQ about the future resource requirements and present state of provisioned resources. The high-level initial requests have a schema that includes mandatory elements like template type of the request {`condor | hadoop | mpi`}[_sp][_storage][_multi], type of request, and optional elements like VM image, bandwidth, storage, stitchport attributes, etc. The adaptation request schema includes the current application view of the available resources, QoS expectations (for eg. number of resources to meet deadline), and other optional elements to guide the adaptation policies used by the Request Manager. Messages conforming to the schema are routed to an AMQP based messaging framework (RabbitMQ [15]). The Request Manager is responsible for publishing essential elements of the slice manifests, which contain the slice state, into the AMQP space so that the ShadowQ is aware of the state of (newly) provisioned or de-provisioned resources.

Parsing and annotation of requests: The Request Manager parses the high-level requests and generates an initial NDL-OWL request template by invoking ndlLib. Sometimes, even though Layer 3 network configuration is the job of the application/user, it is desirable to automate this configuration, especially in the case of on-ramps to external data sources. The Request Manager takes a skeleton of the NIaaS request and fills in the details related to the higher layers (e.g. IP address assignment). When an application needs to extend a slice into a static resource available via a stitchport, it will obtain the stitchport identifier, and pass that information to the Request Manager. The Request Manager is responsible for contacting stitchport registry and mapper, to find out conforming IP address spaces for the VMs in the slice, so that when the slice is instantiated, the resources in the slice are ready to communicate with pieces of infrastructure outside the control of ExoGENI, and with other elements in the slice. The automatic IP assignment also aids scalable experimentation.

Interaction with ExoGENI controller: The ExoGENI controller is an entity that serves client requests to ExoGENI for virtual topologies. The Request Manager interacts with the ExoGENI controller to send the NIaaS resource requests and obtain the slice manifests. It periodically queries ExoGENI to find out when resource provisioning has successfully completed, and then sends out the updated slice manifest to the AMQP space for consumption by the ShadowQ.

4) Resource Adaptation: The Request Manager interacts with one or more pluggable policy modules that decide on when to ask NIaaS for more resources, when to relinquish existing resources, how many resources to ask for while still satisfying the the requirements imposed on the application end by ShadowQ. The create and modify policies act on information from ShadowQ, monitoring data from slices, current provisioning state from ExoGENI, and decide on the actions to take on the slice. These control policies actuate dynamic infrastructure modifications for closed-loop feedback control. The actions include enabling compute elasticity - growing and shrinking compute resource pools. Figure 3 shows the different phases of resource adaptation.

III. Evaluation

In this section, we will present our evaluation of workflow introspection and it's use with Möbius to drive provisioning and adaptation. We will first describe the experiment setup in Section III-A. In Section III-B, we present our results on the accuracy of workflow introspection with ShadowQ. In Section III-C, we present the evaluation of the use of ShadowQ estimates for adaptation of the representative workflow.

A. Experiment Setup

1) Workflow Application - Montage: The experiments used a data-intensive astronomy workflow application as a representative driving example. The workflow is based on the widely used Montage [16] astronomy image application developed at NASA's Infrared Processing and Analysis Center. The Montage workflow is given a region of the sky for which a mosaic is desired, the size of the mosaic in terms of square degrees, and other parameters such as the mission and band to be used. The input images are first reprojected to the coordinate space of the output mosaic, the reprojected images are then background rectified and finally co-added to create the final output mosaic. Figure 4 shows the majority of the tasks and dependencies in the workflow. Montage tasks are single-core, take one or more input images, perform an operation to reproject or combine images, and write an output image.

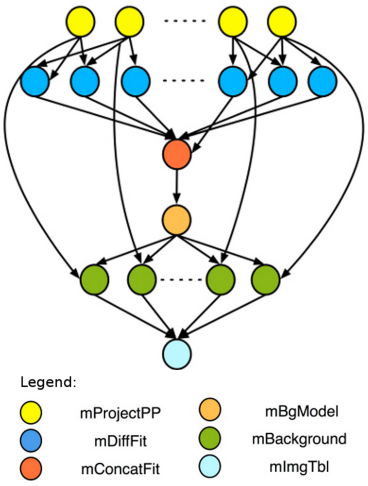

Fig. 4. Montage workflow tasks & dependencies.

2) Infrastructure: We conducted our experiments using two racks from the ExoGENI testbed - an IBM rack at the Florida International University (FIU), Miami, FL, and a Cisco rack at West Virginia Net (WVN), Morgantown, WV. Details about the hardware on these two racks can be found on the ExoGENI wiki [17]. Slices were provisioned from these racks by sending requests for virtual topologies consisting of a set of virtual machines (VM) connected via a broadcast link with a bandwidth of 500 Mb/s. Since Pegasus uses HTCondor, the request to ExoGENI consisted of a HTCondor master VM and a specified number of HTCondor worker VMs connected by links with desired bandwidth, based on `condor` request template. The VM images had pre-requisite software installed like HTCondor and Pegasus. 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. Adaptation actions happened as per Figure 3.

B. Accuracy of ShadowQ Predictions

We conducted experiments to evaluate the accuracy of the simulations used by the ShadowQ. The goal of these experiments is to determine whether the ShadowQ is able to predict the finish time of the workflow, and the start time of the individual jobs, with enough accuracy to be useful. We estimate that the final system would require an accuracy of approximately 5 minutes for the finish time of the workflow, and 2 minutes for the start time of individual jobs within 10 minutes of their actual start time.

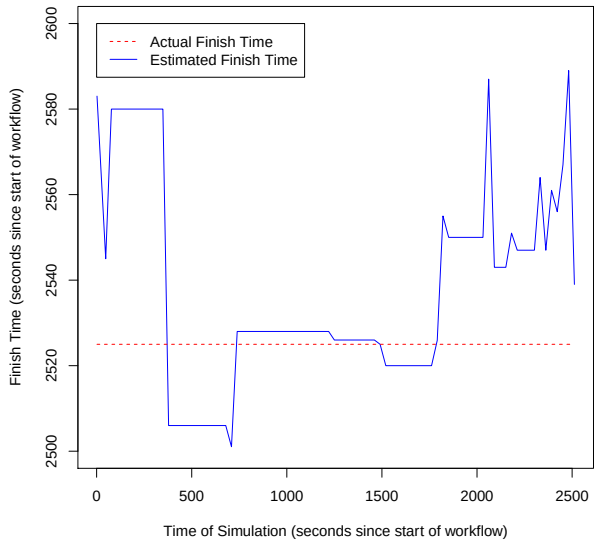

Fig. 5. The majority of estimated workflow finish times predicted by the ShadowQ were within 60 seconds of the actual finish time.

The experiments were conducted on the FIU rack using 1 HTCondor master VM and 4 HTCondor worker VMs. Each worker VM was configured to advertise one Condor job slot. The HTCondor scheduler was configured with a 60 second maximum scheduling interval, and a 20 second minimum scheduling interval, so that idle jobs are matched with available

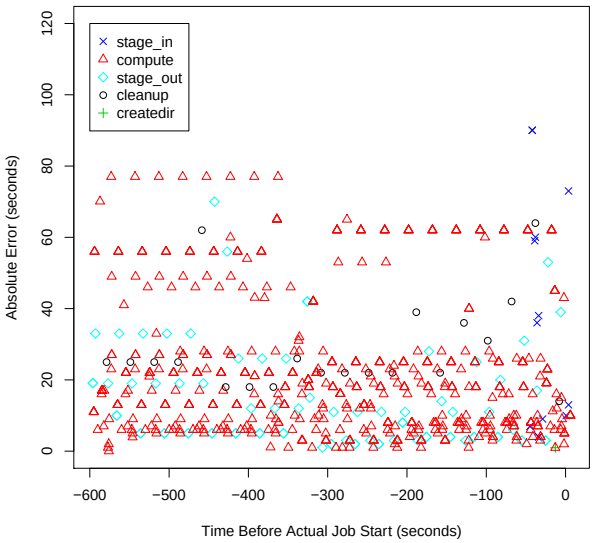

Fig. 6. Job start times predicted by the ShadowQ 10 minutes prior to the actual job start time were within 60 seconds of the actual start for the majority of jobs.

resources every 60 seconds, and when they are queued if it has been more than 20 seconds since the last scheduling cycle. The workflow engine behind Pegasus, DAGMan, has a polling interval of 5 seconds. These values are used as parameters in the ShadowQ simulations.

The experiments used a 4-degree Montage workflow, described in section III-A1, with 3,741 tasks, 1.6 GB of input data, 15 GB of intermediate data, and 3.4 GB of output data. The workflow was clustered based on the number of available job slots, so that each level of the workflow contained no more than 4 jobs. The final, executable workflow contained 47 jobs, including 15 staging jobs, 2 auxiliary jobs, and 30 compute jobs. The critical path of the workflow is approximately 2,500 seconds, and the mean job duration is approximately 143 seconds.

In order for the ShadowQ to simulate the workflow it needs to have estimates of the durations of all the jobs in the workflow and the sizes of input and output files that are transferred. In the production version of this system, those estimates will come from performance models that are automatically derived from historical data. Since these models are not currently available, for the purposes of this evaluation we ran the workflow twice: once to obtain runtime estimates, and then again to evaluate the results of the simulation. In other words, we used the results of the first run to predict the second. Using this approach resulted in total job runtime error in the second run of 5% (sum of absolute errors divided by sum of job runtimes). Since the ShadowQ does not currently include a network model for predicting transfer times given data size, the runtimes of staging jobs were used instead.

In this experiment, the ShadowQ was configured to simulate the remainder of the workflow every 30 seconds for the entire runtime of the workflow. For this workflow each simulation takes much less than 1 second to complete, so frequent simulations are feasible. We expect that production workflows will also have relatively short simulation times to enable frequent re-evaluation of resource requirements.

Figure 5 shows the absolute error for the estimated finish time of the workflow. The error is computed by taking the absolute value of the difference between the finish time of the workflow estimated by the simulations, and the actual finish time of the workflow. For this experiment the error was never more than about 60 seconds for the entire duration of the workflow.

Figure 6 shows the absolute error for the estimated start time of all the jobs in the workflow for the last 10 minutes before they actually started. The error is computed by taking the absolute value of the difference between the start time of the job as predicted by the simulation, and the actual start time of the job. Only data points from the simulations that occurred less than 10 minutes before the actual job start time were included. In all cases, the error was never more than about 90 seconds, and in the majority of cases the error was less than 40 seconds. The relatively large errors for the `stage_in` jobs are caused by the fact that these jobs are treated differently than the others by HTCondor, which is not accounted for in the simulation. Since the `stage_in` jobs occur at the start of the workflow, predicting their start times accurately is less critical than the others. Nonetheless, in the future, we plan to update the simulation to account for this behavior.

The results of these experiments suggest that the simulations done by the ShadowQ are accurate enough to be used for resource provisioning purposes. The simulations were able to accurately predict the finish time of the workflow to within approximately 60 seconds, and the start time of individual jobs to within approximately 40–90 seconds. The accuracy of these predictions depends significantly on the accuracy of the runtime estimates of individual jobs. Since this experiment used runtime estimates that were fairly accurate (total error of 5%), we would expect production workflows with larger runtime estimate errors to have larger simulation errors. Nonetheless, these initial results suggest that the approach is feasible.

C. Using ShadowQ for Adaptation

In this section, we evaluate the use of ShadowQ estimates in adaptation and provisioning of the Montage workflow. These experiments were conducted on the WVN rack. A range of sizes of HTCondor pools were used for these experiments, 1 through 16 HTCondor worker VMs and 1 HTCondor master VM. Each worker VM was configured to advertise one Condor job slot. Hence, for our experiments, the number of slots is equal to the number of HTCondor worker VMs.They used the same 4-degree Montage workflow described in the previous section. We used a workflow clustering of 8 and ShadowQ runtime estimates based on this configuration with 8 HTCondor worker VMs.

Figure 7 shows the workflow makespan (in seconds) using different numbers of HTCondor job slots. These runs were done without ShadowQ and adaptation. Each run with a specific number of VMs was allocated those resources for the entire duration of the run. We observe that the workflow makespan decreases with increasing number of allocated slots, up to 8 slots, and stays constant beyond that. This is

expected because of the chosen clustering factor (8), since the maximum degree of parallelism in the workflow execution is determined by that factor. The purpose of obtaining these workflow makespan values was to determine the minimum possible workflow execution times for each value of slot count. The workflow makespan value using only one VM / slot is particularly important for the following experiments because that value determines the minimum possible "CPU seconds" needed to complete the tasks of the workflow, this case corresponding to 100% utilization of the compute resource.

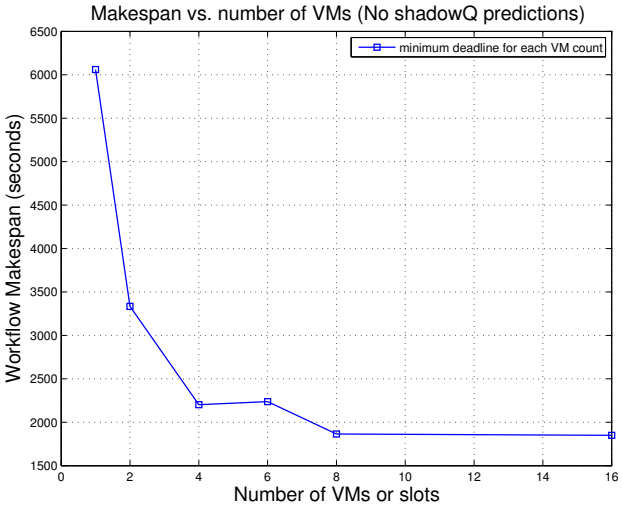

Fig. 7. Workflow makespan vs. number of slots used (no ShadowQ estimates).

From Figure 7, we determined the minimum makespan possible for this workflow from the data point corresponding to 8 slots. Let us refer to it as `min_deadline(8)`. In the following experiment, as an input to ShadowQ, we set the expected deadline (or time to completion) as different factors of `min_deadline(8)`, namely $1.0, 1.25, 1.5, 2.0$, i.e. we wanted to relax the deadline of the workflow and find out whether ShadowQ was able to adjust the resources dynamically as the workflow executes. Let us refer to these factors as different "deadline relaxation factors".

Figure 8 shows the plot of resource usage over workflow execution for different deadline relaxation factors. The X-axis represents seconds from start of the workflow execution, The Y-axis represents the number of slots used at different points of the workflow execution. Note that slots can come and go through provisioning of new HTCondor worker VMs or through de-provisioning of existing HTCondor workers during the workflow execution. Each of the four data plots correspond to a particular deadline relaxation factor, and start with 8 VMs allocated at the start of workflow execution. We observe that, based on the deadline relaxation factors, ShadowQ is able to use its estimates and simulation results to predict resource requirements at different points of the workflow execution, and the Möbius system is able to satisfy the resource requirements as requested by the ShadowQ. We also observe that the resource usage patterns during different points of execution depends on the value of the deadline relaxation factor. For different deadline relaxation factors, ShadowQ is able to adjust the requirements dynamically, thereby minimizing resource usage and maximizing resource utilization, while still meeting the deadlines. The shape of the step graph will also depend on the structure of the workflow and the requirements of the workflow in different phases. This particular example is "top heavy", meaning that a lot more computations happen during the initial parts of the workflow, and since we started with 8 allocated VMs, we only see VMs being de-provisioned as workflow progresses.

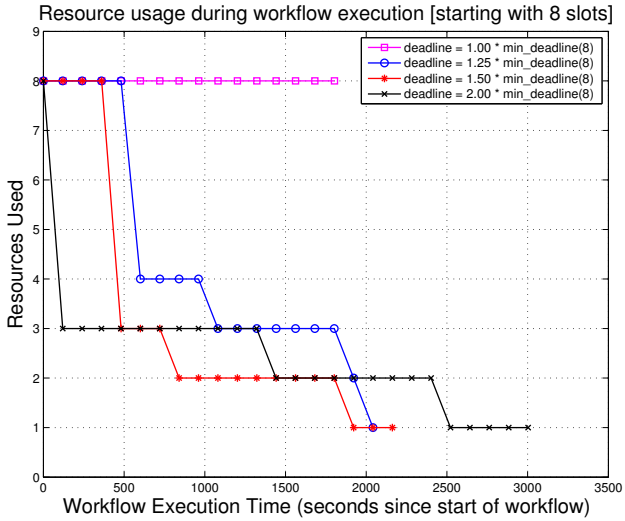

Fig. 8. Resource usage: different deadline factors

The previous result shows that ShadowQ makes resource adjustments happen during workflow execution, but does not show how much better resource utilization actually results from these adjustments. For that, we first need to understand what is the minimum possible resource usage for the given workflow. From figure 7, we know that the maximum resource utilization happens when we are using only one VM during the entire run of the workflow, and the corresponding value of resource usage for this workflow is 6060 "CPU seconds", i.e there is at least 6060 "CPU seconds" (= "CPU seconds(min)") of available compute in this workflow. Let's define "Resource Usage Factor" as the ratio of actual "CPU seconds" used and "CPU seconds(min)". Higher "Resource Usage Factor" implies lower resource utilization.

Figure 9 plots the "Resource Usage Factor" for different deadline relaxation factors. We observe that higher deadline relaxation factors result in lower resource usage up to a point. At the deadline relaxation factor of 1.5, we sacrifice about 50% of the execution time, but can be within 20% of the best resource utilization, and would result in about 50% saving in resource usage. When there are costs associated with resource usage, that is a significant saving when deadline relaxation is possible. With an even higher relaxation factor, we would expect the "Resource Usage Factor" value drop to close to 1. There is always some penalty in the time it takes to provision or de-provision VMs, and the limits of accuracy of ShadowQ predictions, which is why the ratio will never drop to 1. From this result, we can infer that the ShadowQ predictions and the adaptation by the Möbius system offer a unique capability to explore this space. Production workflows would benefit greatly by the determination of this inflection point.

Figure 10 shows the case when the initial allocation

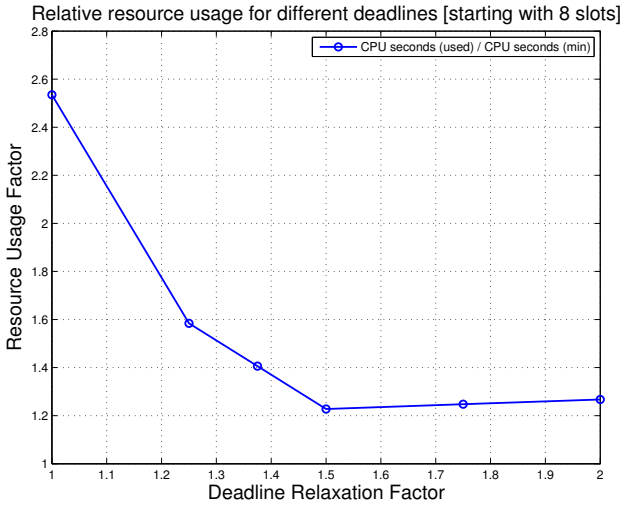

Fig. 9. Resource usage factors for different deadline relaxation factors.

was only for 1 HTCondor worker VM. When we plot the number of resources used over the workflow execution for different deadline relaxation factors, $0.33, 0.5, 0.75$ relative to `min_deadline(1)`, we observe that ShadowQ is able to adjust the resource usage by provisioning and de-provisioning VMs, i. e. it exercises both forms of resource adaptation - addition and deletion of worker VMs.

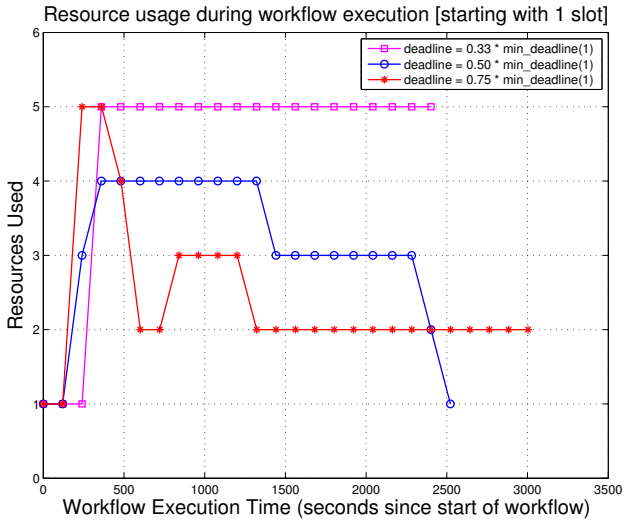

Fig. 10. Resource usage: different deadline factors.

IV. Related Work

There has been considerable work [18], [19] on investigating the effectiveness and applicability of IaaS cloud platforms for executing scientific workflows. Existing workflow engines like Pegasus and Kepler have features [20], [21] that can leverage Amazon EC2 and other cloud platforms for running workflow steps. Researchers have also done cost and performance evaluation of scientific workflows on clouds [22], [23]. There is also existing work on performance analysis and evaluation of cloud infrastructures for scientific computing [24], [25], [26].

A number of public cloud providers (Amazon EC2, Microsoft Azure, Rackspace) offer IaaS abstractions and some ability to orchestrate them together with networks through mechanisms like CloudFormation [27] and Heat [28]. They provide a great deal of resource elasticity. However, their closed nature and difficulty of moving data in, and especially, out of their infrastructure, limit their uses in science applications. Existing cloud research testbeds like FutureGrid [29] are not programmable from a networking perspective. Globus Online [30] project permits users to efficiently move data from one computing resource to another, but doesn't provide unified environments for science workloads.

V. Conclusions and Future Work

We proposed the Möbius controller that facilitates the use of networked clouds for workflow systems like Pegasus. We presented ShadowQ that predicts dynamic resource needs using a novel workflow introspection technique. We showed how these predictions can be used with Möbius to actuate infrastructure adaptation in response to dynamic workflow needs. We also showed that the predictions are fairly accurate and the interplay of prediction and adaptation can balance the performance and resource utilization for the Montage workflow. In the future, we plan to develop and evaluate other prediction outcomes from ShadowQ. We plan to compare the accuracy of the estimates with other non-lookahead predictions. We plan to evaluate other infrastructure adaptation mechanisms, like dynamically adding storage, network links, or changing bandwidth on links.

Acknowledgments

Work for this paper was supported by several grants - NSF CC-NIE (ACI 1245926), DoE ASCR (DE-SC0005286), DoE SciDAC (DE-FG02-11ER26050/DE-SC0006925), NSF SDCI (NSF ACI 1032573), and the NSF GENI (#1872).

References

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

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

[3] I. Baldine, Y. Xin, A. Mandal, P. Ruth, A. Yumerefendi, and J. Chase, "Exogeni: A multi-domain infrastructure-as-a-service testbed," in *8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)*, 2012.

[4] 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.

[5] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 351–364. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465386

[6] Apache MESOS, http://mesos.apache.org/.

[7] D. Irwin, J. S. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum, "Sharing Networked Resources with Brokered Leases," in *Proceedings of the USENIX Technical Conference*, June 2006.

[8] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, and J. Chase, "Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control," in *Supercomputing (SC06)*, November 2006.

[9] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing Energy and Server Resources in Hosting Centers," in *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, October 2001, pp. 103–116.

[10] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 69–82. [Online]. Available: https://www.usenix.org/conference/icac13/technical-sessions/presentation/nguyen

[11] 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*, vol. 13, no. 3, pp. 219–237, 2005.

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

[13] R. Ferreira da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B. Tovar, and M. Livny, "Toward fine-grained online task characteristics estimation in scientific workflows," in *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, ser. WORKS '13, 2013.

[14] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: The Definitive Guide Time to Relax*, 1st ed. O'Reilly Media, Inc., 2010.

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

[16] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M. Su, T. A. Prince, and R. Williams, "Montage a grid portal and software toolkit for science grade astronomical image mosaicking," *Int. J. Comput. Sci. Eng.*, vol. 4, no. 2, pp. 73–87, Jul. 2009. [Online]. Available: http://dx.doi.org/10.1504/IJCSE.2009.026999

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

[18] G. Juve and E. Deelman, "Scientific workflows and clouds," *Crossroads*, vol. 16, no. 3, pp. 14–18, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1734160.1734166

[19] E. Deelman, G. Juve, M. Malawski, and J. Nabrzyski, "Hosted science: Managing computational workflows in the cloud," *Parallel Processing Letters*, vol. 23, no. 2, 2013.

[20] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman, "Experiences using cloud computing for a scientific workflow application," in *Proceedings of the 2nd international workshop on Scientific cloud computing*, ser. ScienceCloud '11. New York, NY, USA: ACM, 2011, pp. 15–24. [Online]. Available: http://doi.acm.org/10.1145/1996109.1996114

[21] J. Wang and I. Altintas, "Early cloud experiences with the kepler scientific workflow system," *Procedia Computer Science*, vol. 9, no. 0, pp. 1630 – 1634, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050912003006

[22] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 22:1–22:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389026

[23] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An evaluation of the cost and performance of scientific workflows on amazon ec2," *J. Grid Comput.*, vol. 10, no. 1, pp. 5–21, 2012.

[24] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 159–168.

[25] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "A factor framework for experimental design for performance evaluation of commercial cloud services." in *CloudCom*. IEEE, 2012, pp. 169–176. [Online]. Available: http://dblp.uni-trier.de/db/conf/cloudcom/cloudcom2012.htmlLiOZC12

[26] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "A performance analysis of ec2 cloud computing services for scientific computing." in *CloudComp*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds., vol. 34. Springer, 2009, pp. 115–131. [Online]. Available: http://dblp.uni-trier.de/db/conf/cloudcomp/cloudcomp2009.html#OstermannIYPFE09

[27] "AWS CloudFormation," http://aws.amazon.com/cloudformation/.

[28] "OpenStack Heat Project," https://wiki.openstack.org/wiki/Heat.

[29] "FutureGrid," https://portal.futuregrid.org/.

[30] I. Foster, "Globus Online: Accelerating and Democratizing Science through Cloud-Based Services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011.