A Service Framework for Scientific Workflow Management in the Cloud

Yong Zhao, *Member, IEEE*, Youfu Li, *Student Member, IEEE*, Ioan Raicu, Shiyong Lu, Cui Lin, *Member, IEEE*, Yanzhe Zhang, Wenhong Tian and Ruini Xue, *Member, IEEE*

Abstract—Cloud computing is an emerging computing paradigm that can offer unprecedented scalability and resources on demand, and is getting more and more adoption in the science community, while scientific workflow management systems provide essential support such as management of data and task dependencies, job scheduling and execution, provenance tracking, etc., to scientific computing. As we are entering into a "big data" era, it is imperative to migrate scientific workflow management systems into the Cloud to manage the ever increasing data scale and analysis complexity. We propose a reference service framework for integrating scientific workflow management systems into various Cloud platforms, which consists of eight major components, including Cloud workflow management service, Cloud resource manager, etc., and 6 interfaces between them. We also present a reference framework for the implementation of Cloud Resource Manager, which is responsible for the provisioning and management of virtual resources in the Cloud. We discuss our implementation of the framework by integrating the Swift scientific workflow management system with the OpenNebula and Eucalyptus Cloud platforms, and demonstrate the capability of the solution using a NASA MODIS image processing workflow and a production deployment on the Science@Guoshi network with support for the Montage image mosaic workflow.

Index Terms—Cloud Workflow, Cloud Resource Management, Reference Service Framework, Swift, Virtual Cluster Provisioning, Workflow-as-a-Service

1 Introduction

SCIENTIFIC workflow management systems (SWFMSs) have been proven essential to scientific computing and services computing [4], [24], [25], [26] as they provide functionalities such as workflow specification, process coordination, job scheduling and execution, provenance tracking, and fault tolerance. Systems such as Taverna [11], Kepler [9], Vistrails [10], Pegasus [8], Swift [32], and VIEW [25] have seen wide adoption in various disciplines such as Physics, Astronomy, Bioinformatics, Neuroscience, Earth Science, and Social Science. Nevertheless, advances in science instrumentation and network technologies are posing great challenges to our workflow systems in both data scale and application complexity.

Industrial and Scientific communities are facing a "data deluge" [7] coming from products, sensors, satellites experiments and simulations. Scientists, manufacturers and developers are attempting multifarious methods to deal with the ever-increasing computing and storage problems arising in the "big data" era. The Large Hadron Collider[1] at CERN can generate more than 100 terabytes of collision data per second; GenBank[2], one of the largest DNA databases, hosts over 120 billion bases and the number is expected to double every 9-12 months. Data volumes are also increasing dramatically in physics, earth science, medicine, and many other disciplines. As for application complexity, a protein simulation problem [29] involves running many instances of a structure prediction simulation, each with different random initial conditions and performs multiple rounds. The number of jobs can easily reach hundreds of thousands, and can run up to tens of CPU years.

As an emerging computing paradigm, Cloud computing [1] is gaining tremendous momentum in both academia and industry: Amazon, Google, IBM, and Microsoft all released their Cloud platforms one after another. Meanwhile, several open source Cloud platforms, such as Hadoop[3], OpenNebula[4], Eucalyptus [28], Nimbus [22], and OpenStack[5], become available with fast growth of their own communities. Scientific workflow systems have been formerly applied over a number of execution environments such as workstations, clusters/Grids, and supercomputers, where the new Cloud computing paradigm with unprecedented size of datacenter-level resource pool and on-demand resource provisioning can offer much more to such systems, enabling scientific workflow solutions capable of addressing peta-scale scientific problems. The benefit of managing and running scientific workflows on top of the Cloud can be multifold:

- *Y. Zhao, Y. Li, W. Tian and R. Xue are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China. (e-mail: yongzh04@gmail.com, youfuli.fly@gmail.com, tian_wenhong@uestc.edu.cn, xueruini@gmail.com).*
- *I. Raicu is with the Department of Computer Science, Illinois Institute of Technology, Chicago IL, USA. (e-mail: iraicu@cs.iit.edu).*
- *S. Lu is with the Department of Computer Science, Wayne State University, Detroit MI, USA. (email: shiyong@wayne.edu).*
- *C. Lin is with the Department of Computer Science, California State University, Fresno, USA. (email: clin@csufresno.edu).*
- *Y. Zhang is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. (email: zhangyanzhe@ict.ac.cn).*

*Please note that all acknowledgments should be placed at the end of the paper, before the bibliography (**note that corresponding authorship is not noted in affiliation box, but in acknowledgment section**).*

[1] http://lhc.web.cern.ch
[2] http://www.ncbi.nlm.nih.gov/genbank
[3] http://hadoop.apache.org/
[4] http://www.OpenNebula.org
[5] http://www.openstack.org

1) The scale of scientific problems that can be addressed by scientific workflows can be greatly increased compared to cluster/Grid environments, which was previously upbounded by the size of a dedicated resource pool with limited resource sharing extension in the form of virtual organizations. Cloud platforms can offer vast amount of computing resources as well as storage space for such applications, allowing scientific discoveries to be carried out in a much larger scale.

2) Application deployment can be made flexible and convenient. With bare-metal physical servers, it is not easy to change the application deployment and the underlying supporting platform. However with virtualization technology in a Cloud platform, different application environments can be either pre-loaded in virtual machine (VM) images, or deployed dynamically onto VM instances.

3) The on-demand resource allocation mechanism in the Cloud can improve resource utilization and change the experience of end users for improved responsiveness. Cloud-based workflow applications can get resources allocated according to the number of nodes at each workflow stage, instead of reserving a fixed number of resources upfront. Cloud workflows can scale out and in dynamically, resulting in fast turn-around time for end users.

4) Cloud computing provides much larger room for the trade-off between performance and cost. The spectrum of resource investment now ranges from dedicated private resources, a hybrid resource pool combining local resource and remote Clouds, and full outsourcing of computing and storage to public Clouds. Cloud computing not only provides the potential of solving larger-scale scientific problems, but also brings the opportunity to improve the performance/cost ratio.

There are various challenges associated with migrating and adapting an SWFMS in the Cloud, including architectural challenges, integration challenges, computing challenges, data management challenges, language challenges and service management challenges. We have discussed the challenges in depth in an early paper [12]. Meanwhile, Security has been identified as one of the main concerns for the adoption and success of the Cloud [1] and is the first major service that needs to be provided by a Cloud provider. In many cases, large simulations are organized as scientific workflows that run on Distributed Computing Infrastructures (DCIs), and we realize that SWFMSs are diverse in many aspects, such as workflow models, workflow languages, workflow engines, and so on. In many cases, one workflow system engine is dependent on one specific DCI, porting a SWFMSs to run on another DCI may cost a large quantity of extra effort. So in practice, researchers may choose to integrate a specific SWFMS into a particular Cloud, whichever takes the minimum effort to migrate.

Taking into consideration all the advantages, requirements and challenges, we propose a service framework for migrating and integrating SWFMSs into various Cloud platforms. Through the introduction of the reference service framework and the implementation of different modules that can be mapped into the proposed framework, we try to achieve three goals: 1) proposing a framework to bridge various SWFMSs with multiple heterogeneous Cloud environments; 2) breaking the limitations that a specific SWFMS is bound to a particular Cloud environment; 3) providing both practical and reference value to researchers who are devoted to the study of running scientific workflows in Clouds, so that they can contribute and share components designed and implemented by the guidance of the service framework, which is beneficial to both the workflow and the Cloud computing communities. The service framework covers all the major aspects involved in workflow management in the Cloud, from the client-side workflow submission to the underlying Cloud resource management. Our major contributions are: 1) we propose a reference service framework for migrating SWFMSs into various Cloud platforms; 2) we also propose a reference service architecture for Cloud resource management that is a core component of the framework; 3) we provide implementations of the service framework by integrating the Swift workflow system with the OpenNebula and Eucalyptus Cloud platforms; 4) we analyze the integration efficiency of our approach in a small cluster based Cloud setting, a public science Cloud platform, and also present a use case of production deployment.

The rest of the paper is organized as follows: in section II, we discuss related work in running scientific applications and workflows in the Cloud. In section III, we present our reference service framework for migrating different SWFMSs into diverse Cloud platforms. In the Implementation Experience section, we discuss our experience in integrating the Swift workflow system into the OpenNebula and Eucalyptus Cloud platforms. In the Experiment section, we demonstrate and analyze our integration using a NASA MODIS image processing workflow and the Montage image mosaic workflow [19], and in the last section, we draw our conclusions and discuss future work.

2 Related Work

There have been a couple of explorers that tried to run workflow on Clouds. The series of works [13], [30] focused on running scientific workflows that are composed of loosely coupled parallel applications on various Clouds. The study conducted on an experimental Nimbus Cloud testbed [14] dedicated to science applications involved a non-trivial amount of computation performed over many days, which allowed the evaluation of the scalability as well as the performance and stability of the Cloud over time. Their studies demonstrated that the multi-site Cloud computing is a viable and effective solution for some scientific workflows, and the networking and management overhead across different Cloud infrastructures do not have a major effect on the overall user experience, and the convenience of being able to scale resources at runtime outweighs such overhead.

The deployment and management of workflows over the current existing heterogeneous and not-yet interoperable Cloud providers, however, is still a challenging task for workflow developers. The series of works [3], [15] presented a broker-based framework to support the execution of workflow applications on a multi-Cloud environment. Wang et al. [16] designed a Workflow as a Service (WFaaS) architecture focused on responding continuous workflow requests and scheduling their executions in the Cloud. After proposing

four heuristic workflow scheduling algorithms for the WFaaS architecture, they analyzed the differences and best usages of the algorithms in terms of performance, cost and the price/performance ratio via experimental studies. Sunflower [5] was an adaptive P2P agent-based framework for configuring, enacting, managing and adapting autonomic workflows on hybrid Grid-Cloud infrastructures. To orchestrate Grid and Cloud services, Sunflower utilized a bio-inspired autonomic choreography model and integrated the scheduling algorithm with a provisioning component that could dynamically launch virtual machines in a Cloud infrastructure to provide on-demand services in peak-load situations.

Approaches for automated provisioning include the Context Broker [22] from the Nimbus project, which supported the concept of "one-click virtual cluster" that allowed clients to coordinate large virtual cluster launches in simple steps. The Wrangler system [23] was a similar implementation that allowed users to describe a desired virtual cluster in XML format, and send it to a web service, which managed the provisioning of virtual machines and the deployment of software and services. It was also capable of interfacing with many different Cloud resource providers.

Upon dynamic resource allocation for scientific workflows, Tram Truong Huu et al. [31] described a framework that automated Cloud resource allocation, deployment and application execution control. It was based on a cost estimation model that took into account both virtual network and nodes managed by the Cloud. Simon Ostermann et al. [21] investigated the usability of compute Clouds to extend a Grid workflow middleware and showed in a real implementation that it could speed up executions of scientific workflows. Elena Apostol et al. [18] addressed key requirements in managing resources at the infrastructure level -- new resources can be dynamically allocated on-demand or policy-based.

The studies mentioned above were either focused on running workflow applications on Clouds or on the deployment and management of integrating workflows into Clouds, dealing with workflow scheduling, resource allocation, application adaptation, performance evaluation, etc., however, a normalized, service-oriented integration framework is still missing. As running scientific workflows as a service in the Cloud platforms involves a variety of systems and techniques, researching and designing of a service-oriented framework can help to standardize the integration procedure and interaction between essential systems, and foster community collaboration.

In this paper, we propose a generic service framework to integrate SWFMSs with various Cloud based DCIs, which covers a wide spectrum from workflow management and migration into Clouds, task scheduling, Cloud resource management, and virtual resource provisioning and recycling. We define a series of interfaces to standardize the interactions between different components. Implementation of different components can be reused and migrated to the service framework according to the interface definition. We also present a reference architecture for the implementation of Cloud Resource Manager, which is a key component in the service framework.

3 Service Framework

In this section, we discuss a service framework for migrating and adapting SWFMSs into various Cloud platforms. Before we go into further details of the service framework, we first discuss some background information with regard to integration options and challenges.

3.1 Integration Options

In an early paper [12], we identified four implementation approaches to the deployment of SWFMSs in a Cloud computing environment according to the reference architecture for SWFMSs [25]. The reference architecture for SWFMSs was proposed as an endeavor to standardize the SWFMS research and development efforts, and an SOA-based instantiation was first implemented in the VIEW system. As shown in Fig. 1, the reference architecture consists of 4 logical layers, 7 major functional subsystems, and 6 interfaces.

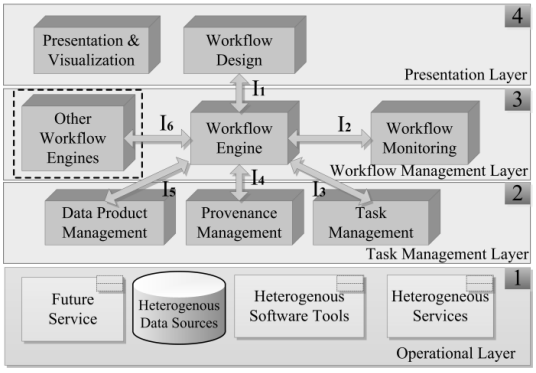

Fig. 1. A Reference Architecture for SWFMSs

The four deployment options, accordingly, correspond to deploying different layers of the reference architecture into the Cloud:

1) *Operational-Layer-in-the-Cloud*. In this solution, only the Operational Layer lies in the Cloud with an SWFMS running out of the Cloud. An SWFMS can now leverage Cloud applications as another type of task components. Cloud-based applications can take advantage of high scalability provided by the Cloud and large resource capacity provisioned by data centers. This solution also relieves a user from the concern of vendor lock-in due to the relative ease of using alternative Cloud platforms for running Cloud applications. However, the SWFMS itself cannot benefit from the scalability offered by the Cloud.

2) *Task-Management-Layer-in-the-Cloud*. Both the Operational Layer and the Task Management Layer will be deployed in the Cloud. The Data Product Management, Provenance Management, and Task Management components can now leverage the high scalability provided by the Cloud. For Task Management, rather than accommodating the user's request based on a batch-based scheduling system, all or most tasks with a ready state can now be immediately deployed over Cloud computing nodes and executed instead of waiting in a job queue for the availability of resources. One limitation of this solution is that the economic cost associated with the storage of provenance and data products in the Cloud. Moreover, although task scheduling and management can benefit from the scalability offered by the Cloud, workflow scheduling and management do not since

the workflow engine runs outside of the Cloud.

3) *Workflow-Management-Layer-in-the-Cloud*. In this solution, the Operational Layer, the Task Management Layer, and the Workflow Management Layer are deployed in the Cloud with the Presentation Layer deployed at a client machine. This solution provides a good balance between system performance and usability: the management of computation, data, and storage and other resources are all encapsulated in the Cloud, while the Presentation Layer remains at the Client to support the key architectural requirement of user interface customizability and user interaction support. In this solution, both workflow and task management can benefit from the scalability offered by the Cloud, but the downside is that they become more dependent on the Cloud platform over which they run.

4) *All-in-the-Cloud*. In this solution, a whole SWFMS is deployed inside the Cloud and accessible via a Web browser. A distinct feature of this solution is that no software installation is needed for a scientist and the SWFMS can fully take advantage of all the services provided in a Cloud infrastructure. Moreover, the Cloud-based SWFMS can provide highly scalable scientific workflows and task management as services, providing one kind of Software-as-a-Service (SaaS). One concern the user might have is the economic cost associated with the necessity of using Cloud on a daily basis, the dependency on the availability and reliability of the Cloud, as well as risks associated with vendor lock-in.

For easy integration with a Cloud platform, a *"Task-Management-layer-in-the-Cloud"* approach can be chosen by implementing, for instance an "Amazon EC2" provider to Swift, then tasks in a Swift workflow can be submitted into EC2 and executed on EC2 VM instances. However, this approach would leave most of the workflow management and dynamic resource scaling outside the Cloud. For application developers, we would like to free them from complicated Cloud resource configuration and provisioning issues, and provide them with the convenience and transparency to scalable Cloud resources, therefore we choose to take the *"Workflow-Management-Layer-in-the-Cloud"* approach, which requires minimal configuration at the client side and supports easy deployment with virtualization techniques.

3.2 Integration Challenges

Many of the immediate challenges to running scientific workflows on the Cloud are to integrate scientific workflow systems with Cloud infrastructure and resources. As we have discussed in the previous section, the degree of integration also depends on how we choose to deploy an SWFMS into Clouds. While we certainly cannot cover all the aspects of the integration problems that we could encounter in the "All-in-the-Cloud" approach, we strive to identify some practical ones, and also discuss possible solutions to them.

Applications, services, tools integration: In the Operational-Layer-in-the-Cloud approach, we treat applications, services, and tools hosted in the Cloud as task components in a workflow, the scheduling and management of a workflow are mostly outside the Cloud, where these task components are invoked as they are scheduled to execute. The invocation would need the right interface to interact with such applications, services, tools, for instance, via HTTP or REST protocols, or Web services calls, and then in the workflow, it needs to transform the output of one invocation, and then feed it as an input to another invocation. A majority of the mashup sites (such as those that leverage Google's map service) take this approach, and they use some *ad hoc* scripts and programs or shimming techniques [17] to glue the services together. An early exploration of the Taverna workflow engine and gRAVI services in the caBIG project [42] can also be thought as an example of integrating an off-the-shelf workflow engine with Cloud/Grid services. gRAVI can rapidly wrap and expose applications, scripts and workflows as Web services, and deploy them into Grid, or the Nimbus Cloud environment.

We notice that while the approach works for applications with small data exchanges, moving large dataset in and out the Cloud would incur serious overhead. For data intensive applications, it is necessary to migrate data into the Cloud. While Amazon's Cloud services allow loading data into the S3 storage, and then having the EC2 computing service access the data stored in S3, the MapReduce style of Cloud services such as Hadoop, would actually require computation to be collocated with storage (the same node that is used for storage is also used for computation) to explore data locality and avoid expensive data movement within and across data centers. Loading data in and out the Cloud is not a trivial process, for instance, within Microsoft, to load each day's Bing search log into the Cloud, this task itself takes hundreds of dedicated servers working around the clock. So in an ideal Cloud workflow solution, we should avoid such operations as much as possible.

Once we decide to get task dispatching and scheduling into the Cloud, resource provisioning becomes the next issue to resolve. Although conceptually Cloud offers uncapped resources, and a workflow can request as many resources as it requires, this comes with a cost and the presumption that the workflow engine can talk directly with the resource allocated in the Cloud (Which is usually not true without tweaking the configuration of the workflow engine). Taking these two factors into consideration, some existing solutions such as Nimbus would acquire a certain number of virtual machines (e.g. EC2 compute nodes), and assemble them as a virtual cluster, onto which existing cluster management systems such as PBS can be deployed and used as a job submission/ execution service that a workflow engine can directly interact with. An existing study [20] simply choose manual deployment over automated provisioning, in which the provisioning step involves construction of a virtual Condor pool, where the VMs act as Condor worker nodes and report to a Condor Master node that runs on a submit host outside the Cloud. This belongs to the Task-Management-Layer-in-the-Cloud approach, and it requires the Condor connection broker to enable VMs with private network addresses to talk to the outside submit host.

Debugging, monitoring, and provenance tracking for workflows become increasingly difficult in the Cloud environment, since compute resources are usually dynamically assigned and based on virtual machine images, the environment that a task is executed on could be destroyed right after the task is finished, and assigned to a complete different user and task. Some Clouds also support task migration

where tasks can be migrated to another virtual machine if there is a problem with the node that the task was running on.

3.3 Service Framework

We propose a structured service framework that addresses the above mentioned challenges and covers all the major aspects involved in the migration and integration of SWFMSs into the Cloud, from client-side workflow specification, service-based workflow submission and management, task scheduling and execution, to Cloud resource management and provisioning. As illustrated in Fig. 2, the service framework includes 4 layers, 8 components and 6 interfaces.

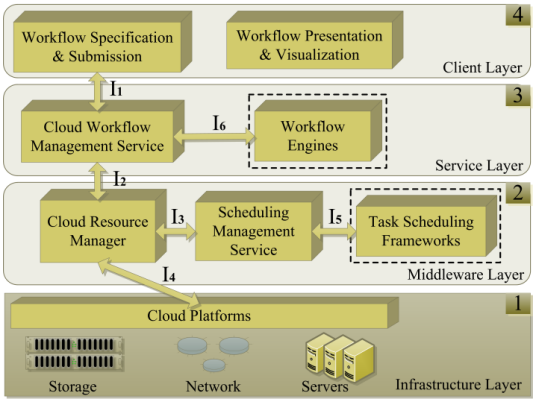

Fig. 2. The Service Framework

3.3.1 Layers

The first layer is the *Infrastructure* Layer, which consists of multiple Cloud platforms with the underlying server, storage and network resources. This layer provides IaaS level support such as the management of the fundamental physical equipment, virtual machines and storage systems to upper layers. The separation of the Infrastructure Layer from other layers isolates the science-focused and technology-independent problem solving environment from the underlying fast advancing high-end computing infrastructure.

The second layer is called the *Middleware* Layer. This layer is responsible for resource management and provisioning, and responding to requests from upper-layer and supporting various scheduling frameworks. All the operations that need to access the underlying resources are encapsulated in this layer. According to the description in the Integration Options section, this layer is responsible for the requirements requested by the *Task-Management-Layer-in-the-Cloud* option. Moreover, the separation of the Middleware Layer from the Infrastructure Layer promotes the extensibility of the Infrastructure Layer with new Cloud platforms and new high-end computing facilities, and localizes system evolution due to hardware or software advances to the interface between the Infrastructure Layer and the Middleware Layer.

The third layer is the *Service* Layer, which is responsible for providing scientific workflow management as a service to the upper clients and realizing the execution and monitoring of scientific workflows. This layer also provides interfaces to support various workflow engines. According to the integration options, the Service Layer fulfills the requirements addressed in the *Workflow-Management-Layer-in-the-Cloud* option. The separation of the Service Layer from the Middleware Layer concerns two aspects: 1) it isolates the choice of a workflow model from the choice of a task model, so changes to the workflow structure do not need to affect the structures of tasks and 2) it separates workflow scheduling from task execution, thus provides space for performance and scalability of the whole management system.

The fourth layer is the *Client* Layer, which provides the functionality of workflow design, specification, visualization and various user interfaces and tools for workflow submission, resource configuration etc. The Client layer may be out of the Cloud to circumvent the disadvantages discussed in the *All-in-the-Cloud* option. The separation of the Client Layer from other layers provides the flexibility of customizing the user interfaces of the system and promotes the reusability of the rest of system components for different scientific domains.

3.3.2 Subsystems

The eight major functional subsystems correspond to the key functionalities required for workflow management as a service in the Cloud. Although the reference framework may allow the introduction of additional subsystems and their features in each layer, this paper only focuses on the major subsystems and their essential functionalities.

The *Workflow Specification & Submission* subsystem is responsible for producing workflow specifications represented in a workflow specification language that supports a particular workflow model, and the submission of workflows to the Cloud Workflow Management Service subsystem. The Workflow Specification & Submission subsystem may provide users with a standalone or Web-based workflow designer, which may support both graphical- and scripting-based design interfaces, and a workflow submission component to submit workflows. The interoperability of workflows should be addressed in this subsystem by the standardization and conversion of workflow languages.

The *Workflow Presentation & Visualization* subsystem is important especially for data-intensive and visualization-intensive scientific workflows, in which the presentation of workflows and visualization of various data products and provenance metadata in multi-dimensions are key to gaining insights and knowledge from large scale of data and metadata.

The *Cloud Workflow Management Service* subsystem acts as an intermediary between the workflow client and the backend Cloud Resource Manager, and is the key service in the service framework provided to researchers interested in using Cloud-based scientific workflow. It supports the following functionalities: workflow language compilation, workflow scheduling, resource acquisition, and status monitoring. In addition, the implementation of fault-tolerance mechanism can also be defined in the service.

The *Workflow Engines* subsystem supports various workflow engines and can be specified by end-users from the Workflow Specification & Submission subsystem. A workflow engine is the heart of a workflow system and responsible for creating and executing workflow runs according to a workflow run model, which defines the state transitions of each scientific workflow and its constituent task runs. A

workflow run consists of a coordinated execution of tasks, each of which is called a task run. The interoperability of workflows should be addressed by the standardization of interfaces, workflow models, and workflow run models, so that a scientific workflow or its constituent sub-workflows can be scheduled and executed in multiple Workflow Engines that are provided by various vendors.

The *Cloud Resource Manager* (CRM) subsystem is a resource management framework that bridges task scheduling frameworks with various Cloud platforms, such as Amazon EC2, OpenNebula, Eucalyptus, CloudStack, etc. A reference architecture of Cloud Resource Manager and description in detail will be given in the following section.

The *Scheduling Management Service* subsystem is a framework that bridges Cloud Resource Manager with various Task Scheduling Frameworks. It provides a set of operations for the deployment and management of various scheduling frameworks according to configurations specified by users.

The *Task Scheduling Frameworks* subsystem consists of multiple scheduling frameworks, such as Falkon[27], Sparrow, Gearman, and so on, and the framework can be specified by end-users through configuration. It is devised to schedule tasks delivered from the Workflow Engines subsystem.

The *Cloud Platforms Subsystem* refers to various supported Cloud platforms in general and the functionalities can be summarized from the Infrastructure Layer.

3.3.3 Interfaces

In the reference architecture, six interfaces are explicitly defined, which show how each subsystem interacts with other subsystems. The interoperability between the subsystems should be addressed by standardizing the interfaces provided by each subsystem.

Interface I_1 provides a set of interfaces for the communication between Workflow Specification & Submission subsystem and the Cloud Workflow Management Service, so workflow specifications created by workflow design tools can be submitted to a workflow execution environment for compiling, scheduling, and management. Interface I_2 provides a series of interfaces for Cloud Workflow Management Service to interact with Cloud Resource Manager: the Cloud Workflow Management Service sends resource request to allocate specified cluster resources, and the Cloud Resource Manager replies with the cluster information for task execution. Interface I_3 provides a series of interfaces for the Cloud Resource Manager to communicate with the Scheduling Management Service: upon the specified resource requests from Cloud Workflow Management Service are received, the Cloud Resource Manager provisions resources and deploys the user-specified Task Scheduling Framework into the cluster based on the services provided by the Scheduling Management Service, then sends cluster information back to the Cloud Workflow Management Service. Interface I_4 provides a set of interfaces for the Cloud Resource Manager to interact with underlying Cloud Platforms, mostly for resource provisioning, monitoring and recycling. Interface I_5 provides a series of interfaces for the Scheduling Management Service to interact with Task Scheduling Frameworks subsystem: the supported operations upon scheduling frameworks are defined here. Interface I_6 provides a set of interfaces to interoperate with deployed Workflow Engines. Workflow Specifications can be passed through to default or user-specified workflow engine for execution.

3.3.4 Discussion

The motivation of our work is to break through workflows' dependence on the underlying environment, and take advantage of the scalability and on-demand resource allocation of the Cloud. We present a layered service framework for the implementation of integrating SWFMSs into manifold Cloud platforms, which can also be applicable when deploying a workflow system in Grid environments. The separation of each layer enables abstractions and different independent implementations for each layer, and provides the opportunity for scientists to develop a stable and familiar problem solving environment where rapid technologies can be leveraged but the details of which are shielded transparently from the scientists who need to focus on science itself. The Interfaces defined in the framework is flexible and customizable for scientists to expand or modify according to their own specified requirements and environments.

3.4 A Reference Service Architecture for CRM

The Cloud Resource Manager (CRM) is a key module in the Service framework, which is responsible for supporting various underlying Cloud Computing Infrastructures. Introducing the reference architecture for CRM can contribute to the standardization of CRM implementation in the proposed service framework and achieve the reusability of resource management modules. Application developers do not need to implement different resource management modules on different IaaS Cloud platforms.

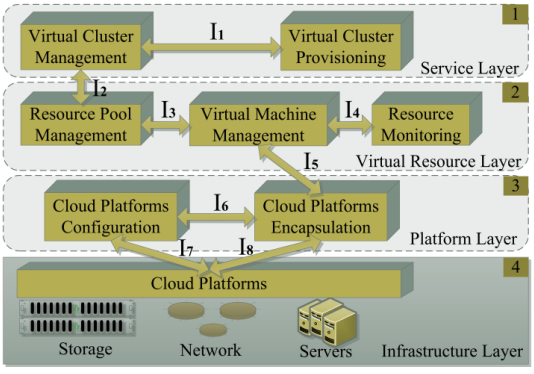

Fig. 3. A Reference Architecture for CRM

The CRM provides scientific workflows with Cloud resource provisioning as a service and the workflows can benefit from the scalability offered by the Cloud. Meanwhile, the dependency on Cloud platforms can be reduced as implementations for various Cloud platforms can be provided, ranging from commercial to open source ones, including Amazon EC2, OpenNebula, Eucalyptus, CloudStack, etc. The architecture, illustrated in Fig. 3, consists of four layers which are loosely coupled between each other.

3.4.1 Layers

The first layer is the *Infrastructure* Layer, which is consistent with the Infrastructure Layer in Fig. 2.

The *Platform* Layer is responsible for the encapsulation

and configuration of underlying Cloud platforms. The separation of Platform Layer from the Infrastructure Layer prevents the changes of underlying Cloud platforms from influencing the implementation of upper functionalities and promotes the extensibility of the Infrastructure Layer.

The *Virtual Resource* Layer is designed to provide efficient and systematic management of virtual resources, including computing resource and storage resource in multiple Cloud platforms. Focusing on virtual resource management, the implementation of this layer can be regarded as further abstraction of the Platform Layer. The separation of the Virtual Resource Layer from the Platform Layer concerns two aspects: 1) it prevents the operations in this layer from accessing the Cloud platforms directly; 2) it isolates resource management from resource instantiation and reduces the degree of coupling between the two layers.

The *Service* Layer, based on all the underlying layers, is responsible for providing cluster resource services according to the task execution service specified by the user. The separation of the Service Layer from other layers provides the flexibility of service customization and extendibility of service update and release.

3.4.2 Components

The eight major functional components implement the requirements of virtual cluster resource provisioning as a service to scientific workflows. In the reference architecture, we focus on major functionalities and extensibility. It may allow introduction of new components to meet extra requirements.

The *Virtual Cluster Provisioning* component accepts resource requests from the Cloud Workflow Management Service and is in charge of provisioning virtual clusters dynamically to the workflow service. All the clusters can be maintained through the *Virtual Cluster Management* component, which may support the alteration of cluster size, cluster status monitoring, and cluster resource recycling and reuse. Both the Virtual Cluster Provisioning and Virtual Cluster Management components support manifold cluster types, including single-server cluster, multi-server cluster and peer-to-peer distributed cluster.

The *Virtual Machine Management* component defines a series of interfaces upon the operations of virtual machines, such as launch, reboot, termination, etc. Based on the Virtual Machine Management component, the *Resource Pool Management* component manages all the launched VM instances as a pool, and will apply for more resources if there is a shortage of virtual machines in the pool. The *Resource Monitoring* component may consist of monitoring and heartbeat modules which are respectively in charge of monitoring CPU, memory and IO, and checking the heartbeat of each instance.

To provide the basic management for multiple Cloud platforms, we implement the *Cloud Platforms API Encapsulation* component, which is responsible for the API encapsulation of all the supported platforms by interacting with the underlying Infrastructure Layer. The configuration information of the platforms, which is required to initialize the virtual resource management, can be edited and read in XML format files through the functions offered by the *Cloud Platforms Configuration* component.

3.4.3 Interfaces

In the reference architecture for CRM, eight interfaces are explicitly defined, which show how each component communicates with other components. The interoperability between components should be addressed by standardizing the interfaces provided by each component. The details of the interfaces between components at the same layer are not shown in the figure for simplicity.

Interface I_1 provides a series of interfaces for the interaction between the Virtual Cluster Management and Virtual Cluster Provisioning components, so the required cluster reference information can be sent between the two components and the provisioned clusters can be recycled and reused. Interface I_2 provides a set of interfaces for the Virtual Cluster Management component to communicate with the Resource Pool Management component, and the resource reference information can be transmitted through these interfaces. Interface I_3 defines a set of interfaces for the communication between Resource Pool Management and Virtual Machine Management, so we can invoke the functions encapsulated in Virtual Machine Management to manage the virtual machines in the resource pool. Interface I_4 provides a set of interfaces for Virtual Machine Management to interact with the Resource Monitoring component. Through these interfaces, the monitoring information can be used to better manage the virtual machines and respond to exception status. Interface I_5 provides a series of interfaces for the Virtual Machine Management component to invoke the Cloud platform APIs encapsulated in Cloud Platforms Encapsulation and apply for more virtual resources from underlying Cloud platforms. Interface I_6 provides a set of interfaces for the Cloud Platforms Encapsulation component to acquire the specified or user defined Cloud platform configuration from Cloud Platforms Configuration component. Interface I_7 offers a series of interfaces for the Cloud Platforms Configuration component to fetch and modify the configuration of Cloud platforms in the Infrastructure Layer. Interface I_8 defines a set of interfaces for the Cloud Platforms Encapsulation component to access Cloud Platforms to acquire resources, including virtual machines, network and storage resources etc.

4 Implementation Experience

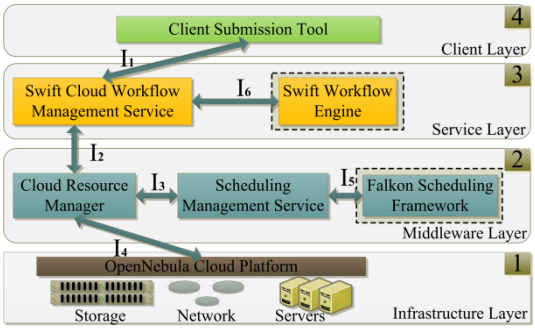

Fig. 4. The Integration Architecture

In this section we describe our experience in integrating the Swift scientific workflow management system [32] with different Cloud platforms based on the service framework we introduced above. The integration supports workflow speci-

fication and submission, on-demand virtual cluster provisioning, high-throughput task scheduling and execution, and scalable resource management in the Cloud. We implement for both the OpenNebula and the Eucalyptus platforms and we show the integration architecture for OpenNebula in Fig. 4.

4.1 Components and Interfaces

As the implementation of service framework involves a variety of systems and techniques, for the purpose of clarity, we list the subsystems, corresponding to Figure 2, in Table 1. And we point out which subsystems are directly from the original systems and which are implemented for the integration. We also define a series of interfaces to standardize the complicated interactions between different essential subsystems. We list the key interfaces, corresponding to Figure 2, in Table 2, and point out the implementation status and interaction relationships. Further details about these interfaces are available at our website[6].

TABLE 1
SUBSYSTEMS IMPLEMENTATION DESCRIPTION

Components	Description	Subsystems
OpenNebula /Eucalyptus	reuse	Cloud Platforms ***(Abbr. CP)***
Falkon Scheduling Framework	minor revision	Task Scheduling Frameworks ***(Abbr. TSF)***
SMS	implemented	Scheduling Management Service ***(Abbr. SMS)***
CRM	implemented	Cloud Resource Manager ***(Abbr. CRM)***
Swift System	minor revision	Workflow Engines ***(Abbr. WE)***
CWMS	implemented	Cloud Workflow Management Service ***(Abbr. CWMS)***
Client Submission Tool	implemented	Workflow Specification & Submission ***(Abbr. WSS)***

TABLE 2
INTERFACES IMPLEMENTATION DESCRIPTION

Interfaces	Description	Interaction
I_1 in Fig 2	implemented	WSS & CWMS
I_2 in Fig 2	implemented	CWMS & CRM
I_3 in Fig 2	implemented	CRM & SMS
I_4 in Fig 2	implemented	CRM & CP
I_5 in Fig 2	under evaluation	SMS & TSF
I_6 in Fig 2	implemented	CWMS & WE

In Table 1 and Table 2, the "reuse" description represents we directly reuse the available components for integration, and "minor revision" means we reuse the available components after modification. The "implemented" description indicates we implement the components and interfaces, including specification, design, development and test. At last, the "under evaluation" description represents those interfaces have been defined and need further adjustment and evaluation for detailed implementation.

In Table 3, we present some interfaces defined in the implementation. To submit a workflow from the client side tool, we define an interface to standardize the submission of workflows, which can be mapped into Interface I_1 in Fig 2. We also list a set of operations corresponding to Interface I_2 in Fig 2, for the interaction between Cloud Workflow Management Service and Cloud Resource Manager, such as sending a cluster request, querying cluster information and releasing a cluster after execution. We have published the definitions of the interfaces between different subsystems at our website and made the code public[7].

TABLE 3
INTERFACES DESIGNED IN IMPLEMENTATION

```
public interface IWorkflowSubmission {
    public boolean submitWorkflow(WorkflowSpecification workflow, ExecutionConfiguration config) throws Exception;
    public WorkflowStatus queryWorkflowStatus(String workflowID) throws Exception;
    public WorkflowResult queryWorkflowResult(String workflowID) throws Exception;
    public boolean retractSubmission(String workflowID) throws Exception;
...}
```

```
public interface IVirtualClusterRequest {
    public VirtualCluster requestCluster(int clusterSize, ClusterDetails details) throws Exception;
    public VirtualCluster queryClusterInformation(String clusterID) throws Exception;
    public boolean releaseCluster(String clusterID) throws Exception;
...}
```

4.2 Infrastructure Layer Implementation

At the infrastructure layer, OpenNebula manages Cloud datacenter resources such as servers, network and storage. The reason we first choose OpenNebula for our implementation is because it has a flexible architecture and is easy to customize.

The OpenNebula Cloud Platform

OpenNebula is a fully open-source toolkit to build private, public and hybrid IaaS Clouds, and a modular system that can implement a variety of Cloud architectures and interface with multiple datacenter services. It orchestrates storage, network, virtualization, monitoring, and security technologies to deploy multi-tier services [6] as virtual machines on distributed infrastructures. The OpenNebula internal architecture can be divided into three layers: Drivers, Core and Tools.

4.3 Middleware Layer Implementation

At the middleware layer, a few components are integrated seamlessly to bridge the gap between the service layer and the underlying infrastructure layer. The components include the Cloud Resource Manager, the Scheduling Management Service, and the Falkon scheduling framework [27]. The Cloud Resource Manager receives resource requests from the Cloud Workflow Management Service and in turn provisions a virtual cluster on-demand with the Falkon scheduling framework deployed into the cluster for high-throughput task scheduling and execution.

The Cloud Resource Manager

Referring to the service architecture of the Cloud Resource Manager we introduced above, our implementation provides support for both the OpenNebula and Eucalyptus platforms. Other Cloud platforms can also be easily mapped into the architecture with the interfaces we define.

[6] http://www.cloud-uestc.cn/projects/serviceframework/index.html.

[7] https://github.com/YoufuLi/Cattles

The following process describes the interaction between each component and the steps to start a Falkon virtual cluster:

1) The *Virtual Cluster Provisioner (VCP)* provides a service interface for the Cloud Workflow Management Service, the latter makes a resource request to VCP.
2) The *Resource Pool Management (RPM)* component initializes and maintains a pool of virtual machines, and a monitoring service based on Ganglia is started on each virtual machine to monitor CPU, memory and IO.
3) Upon a resource request from the workflow service:
   a) The Virtual Cluster Management (VCM) component fetches required number of VMs from the VM pool and interacts with the Scheduling Management Service to deploy the Falkon Scheduling Framework in the cluster:
      i) start the Falkon service in one VM and the Falkon workers in the other VMs.
      ii) make those workers register to the Falkon service.
   b) If the VMs in the pool are not enough, then RPM will make resource request to the underlying OpenNebula platform to create more VM instances.
4) VCP returns the end point reference of the Falkon server to the workflow service, and the workflow service can now dispatch tasks to the Falkon scheduling framework.
5) VCM starts the *Cluster Monitoring Service* to monitor the health of the Falkon virtual cluster.
6) Note that we also implement an optimization technique to speed up the Falkon virtual cluster creation. When a Falkon virtual cluster is decommissioned, we change its status to "standby", and it can be re-activated.
   When VCP receives resource request from the workflow service, it checks if there is a "standby" Falkon cluster, if so, it will return the information of the Falkon service directly to the workflow service, and also checks the number of the Falkon workers already in the cluster.
   a) If the number is more than requested, then the surplus workers are de-registered and put into the pool.
   b) If the number is less than required, then VMs will be pulled from the VM pool to create more workers.

As for the management of VM images, VM instances, and VM network, the Virtual Machine Management component interacts with and relies on the underlying Platform Layer, which is responsible for interacting with the OpenNebula Cloud platform. Our resource provisioning approach takes into consideration not only the dynamic creation and deployment of a virtual cluster with a ready-to-use execution service, but also efficient instantiation and re-use of the virtual cluster, as well as the monitoring and recovery of the virtual cluster.

The Scheduling Management Service

Before sending cluster reference information to the Cloud Workflow Management Service, the Cloud Resource Manager will first interact with the Scheduling Management Service to check the deployment of task scheduling framework, and start scheduling service. We have already implemented the deployment, management and maintenance of Falkon scheduling framework and the other scheduling frameworks can also be easily mapped into the architecture.

The Falkon Execution Service

Falkon is a light-weight task execution framework for optimized task throughput and resource efficiency delivered by a streamlined dispatcher, a dynamic resource provisioner, and the data diffusion mechanism [27] to cache datasets in local disk or memory and dispatch tasks according to data locality. The key design of Falkon is to enable efficient dispatch and execution of large number of small tasks.

4.4 Service Layer Implementation

At the service layer, a Cloud Workflow Management Service based on the Swift workflow management system is presented as a gateway to the Cloud platform underneath. The Cloud workflow management service accepts workflow submissions from the client tool, and makes resource requests to the Cloud Resource Manager.

The Swift Workflow Management System

Swift is a system that bridges scientific workflows with parallel computing. Swift takes a structured approach to workflow specification, scheduling, and execution. It consists of a simple scripting language called SwiftScript for concise specification of complex parallel computations based on dataset typing and iterations [32], and dynamic dataset mappings for accessing large-scale datasets represented in diverse data formats. The Swift system architecture consists of four major components: Program Specification, Scheduling, Execution, and Provisioning, as illustrated in Fig. 5. The reason that we choose Swift as the SWFMS for implementation is because the four major components of the Swift system can be easily mapped into the four layers in the SWFMSs reference architecture and it can provide flexible interfaces for implementation.

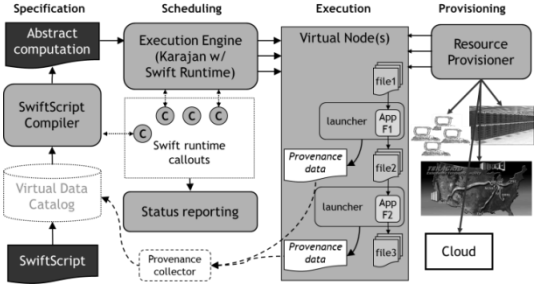

Fig. 5. The Swift System Architecture

Resource provisioning in Swift is very flexible, tasks can be scheduled to execute on various resource providers, where the provider interface can be implemented as a local host, a cluster, a multi-site Grid, or the Amazon EC2 service. In contrast to Cloud environment, running workflows in traditional infrastructures are facing a series of obstacles when dealing with big data problems, including resource provisioning, collaboration in heterogeneous environments, etc. To leverage the unprecedented scalability and resources on demand offered by the Cloud, we encapsulate a wrapper service over the original Swift system, namely the Swift Cloud Workflow Management Service, to interact with client submission and CRM.

The Swift Cloud Workflow Management Service

The Swift Cloud workflow management service acts as an intermediary between the workflow client and the backend Cloud Resource Manager. The service has a Web

interface for configuration of the service, the resource manager and application environments. It supports the following functionalities: SwiftScript programming, SwiftScript compilation, workflow scheduling, resource acquisition, and status monitoring. In addition, the service also implements fault-tolerance mechanism.

4.5 Client Layer Implementation

At the client layer, we provide a client-side development and submission tool for application specification and submission. **The client submission tool** is a standalone java application that provides an IDE for workflow development, and allows users to edit, compile, run and submit SwiftScripts. Scientists and developers can write their scripts in this environment and also test run their workflows on local host, before they make final submissions to the Swift Cloud service to run and monitor workflow execution status.

5 EXPERIMENT

In this section, we show our experiment results for our implementation for both the OpenNebula and Eucalyptus platforms to demonstrate the practicability and capability of the service framework. As the Cloud Resource Manager is the key module in the implementation of service framework, we conduct a series of experiments focused on the capability and efficiency of Cloud Resource Manager and use an image processing workflow to verify the integration.

5.1 OpenNebula Experiments

We demonstrate and analyze the integration implementation in Fig. 4, Section IV using a NASA MODIS image processing workflow. The NASA MODIS dataset[8] we use is a set of satellite aerial data blocks, each block is of size around 5.5MB, with digits indicating the geological feature of each point in that block, such as water, green land, urban area, etc. Details of the experiment can be found in an early paper [2], and we present some of the results to show the applicability of the implementation to a real workflow use case, the efficiency of the cluster recycling mechanism, and the tradeoff between scalability vs. resource provisioning overhead.

5.1.1 MODIS Image Processing Workflow

The workflow (illustrated in Fig. 6) takes a set of such blocks, gets the size of the urban area in each of the blocks, analyzes and picks the top 12 of the blocks that have the largest urban

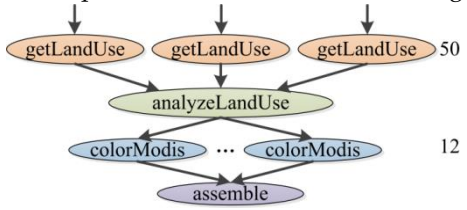

Fig. 6. The MODIS Image Processing Workflow

area, converts them into displayable format, and assembles them into a single PNG file.

[8] http://modis.gsfc.nasa.gov/

5.1.2 Experiment Configuration

We use 6 machines in the experiment, each configured with Intel Core i5 760 with 4 cores at 2.8GHZ, 4GB memory, 500GB HDD, and connected with Gigabit Ethernet LAN. The configuration for each VM is 1 core, 1.5GB memory, 20GB HDD, and we use KVM as the hypervisor. One of the machines is used as the frontend which hosts the workflow service, the CRM, and the monitoring service. The other 5 machines are used to instantiate VMs, and each physical machine can host up to 2 VMs, so at most 10 VMs can be instantiated in the environment. We use the controlled small set of resources in order to reach resource limit easily, they are enough for the other performance tests nonetheless.

5.1.3 Experiment Results

In our implementation, we support dynamic resource provisioning by interacting with the underlying Cloud platforms. In the experiments we would like to measure the benefit of cluster recycling, therefore we pre-instantiate the VMs and put them in the VM pool so that the instantiation overhead will not be counted towards the evaluation results and make them comparable across different underlying Cloud platforms. The time to instantiate a VM is around 42s and this does not change much for all the VMs created.

The serial submission experiment

In this experiment, we first measure the base line for server initialization time and worker registration time. We create a Falkon virtual cluster with 1 server, and varying number of workers, and we don't reuse the virtual cluster.

In Fig. 7, we can observe that the server initialization time

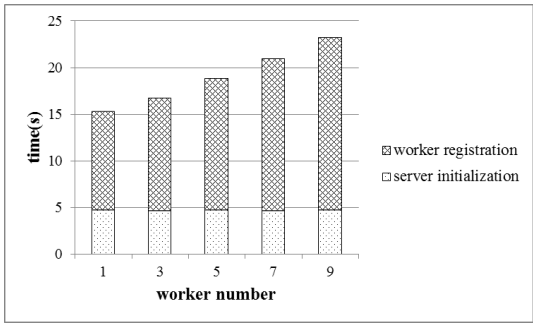

Fig. 7. The Base Line for Virtual Cluster Creation

Fig. 8. Serial Submission, Decreasing Resource Required

is quite stable, around 4.7s every time, and for worker parallel registration, the time increases slightly with the worker number.

Then, we submit a workflow after the previous one has finished to test virtual cluster recycling. In Fig. 8, the re-

sources required for the workflows are one Falkon server with 5 workers, one server with 3 workers and one server with 1 worker. As the workers and server of a "standby" cluster can be reused in the following ones, we can see that for the second and third submissions, the server initialization time is zero, only the surplus workers need to deregister themselves.

Different number of data blocks experiment

In this experiment, we change the number of input data blocks from 50 blocks to 25 blocks, and measure the execution time with varying number of workers in the virtual cluster.

In Fig. 9, we can observe that with the increase of the number of workers, the execution time decreases accordingly (i.e. execution efficiency improves), however at 5 workers to process the workflow, the system reaches efficiency peak. After that, the execution time goes up with more workers. This means that the improvement cannot subsidize the management and registration overhead of the added worker. The time for server initialization and worker registration remain unchanged when we change the input size (as have been shown in Fig. 7). The experiment indicates that while our virtual resource provisioning overhead is well con-

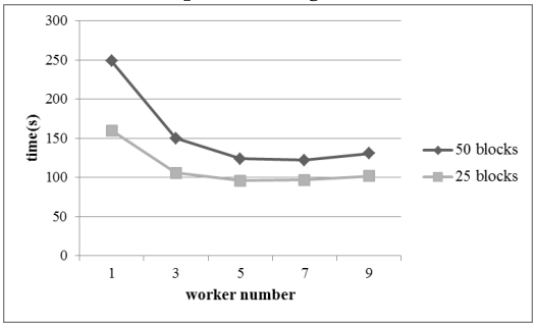

Fig. 9. Different Input Sizes

trolled, we do need to carefully determine the number of workers used in the virtual cluster to achieve resource utilization efficiency.

5.2 Eucalyptus Experiments

In this section, we show the results of using Eucalyptus instead of OpenNebula for resource provisioning. Considering the efficient and convenient service provided by the FutureGrid[9], we choose Eucalyptus for the implementation and deployment. FutureGrid is a project led by Indiana University and funded by the National Science Foundation (NSF) to develop a high-performance Grid test bed that lets scientists collaboratively develop and test innovative approaches to parallel, Grid, and Cloud computing. In addition, the Eucalyptus API is compatible with Amazon EC2 so the implementation can easily support Amazon EC2 Cloud. We measure the performance to establish a baseline for resource provisioning and Cloud resource management overhead in the science Cloud environment.

5.2.1 Experiment Configuration

The instance type used in our experiment is m1.small: 1 CPU Unit, 1 CPU Core and 500MB Memory. All the instances use Ubuntu Server 12.04 as the operating system. In Eucalyptus environment, we also pre-instantiate 32+1 instances and put them in the VM pool to make the evaluation results more intuitive and comparable.

5.2.2 Framework Overhead Evaluation

In the overhead evaluation experiment, we measure the server initialization time and worker registration time to compare with those in the OpenNebula setting.

In Fig. 10, we observe the time to create a Falkon server and start the service is around 11s, much longer than that in

Fig. 10. The Base Line for Virtual Cluster Creation

Fig. 7. We attribute this to the m1.small configuration. The overall time increases slightly with the worker number as all the worker registration is executed concurrently, which shows a similar pattern to that in Fig. 7.

Then we measure the recycling mechanism by submit requests with exponentially decreasing worker number. The "standby" virtual cluster can be reused in the following request. Except the first request, the server initialization time

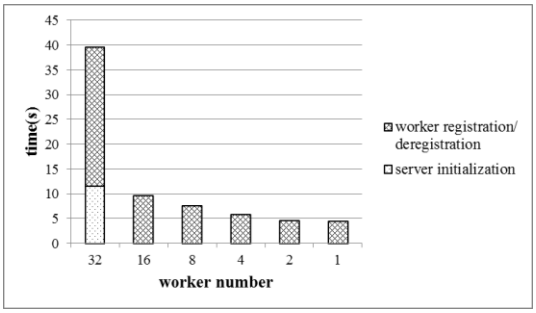

Fig. 11. Serial Submission, Decreasing Resource Required

of the other requests is zero, and the time taken is to deregister 16 workers→8 workers→4 workers→2 workers→1 worker. The results are shown in Fig. 11. We can see the cluster creation time also decreases accordingly.

In Fig. 12, we measure the server initialization and work-

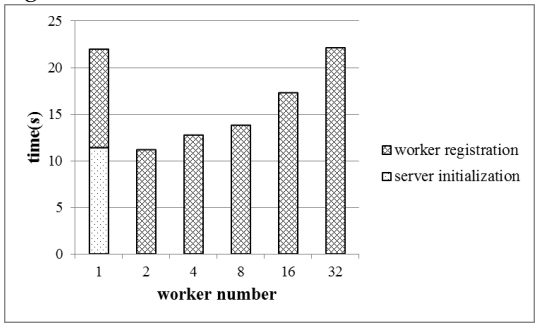

Fig. 12. Serial Submission, Increasing Resource Required

er registration time of a Falkon cluster starting from one

[9] FutureGrid: https://portal.futuregrid.org/

server and one worker. Then we expand the cluster size exponentially by adding 1 worker→2 workers→4 workers→8 workers→16 workers into the cluster.

As shown in Fig. 12, we can see that although the worker number increases exponentially, the time rises almost linearly. The reason is the workers can simultaneously register to the already existing server. The time cost to register each individual worker is similar to that in Fig. 10.

In Fig. 13, we first request a virtual cluster with 1 server and 32 workers, we then make 5 parallel requests for virtual clusters with 1 server and 5 workers. According to the cluster reuse mechanism, one of the clusters can be created based on the available cluster, while the other 4 are created on-demand. In this case, it is much faster to de-register the surplus workers than to create the server from scratch.

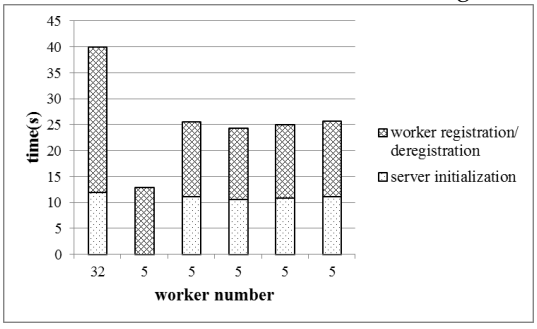

Fig. 13. Parallel Submission, Mixed Resource Required

6 Production Deployment

Besides evaluating the service framework and integration performance in experimental environment, we also deploy the implementation described in Section IV to the Science@Guoshi (Fruit) Network. The deployment in production environment shows our proposed framework is capable of providing a "big data" solution upon data-intensive application for both researchers and engineers.

The Science@Guoshi project[10] is based on Guoshi Network, which is a new media service platform created to promote public learning and learning oriented communities by Chinese Ministry of Education and China Education TV Station. Through the early stage construction and operation, Guoshi Network already has rich functionality and powerful infrastructure service capability, and has gained experience in platform construction, operation and maintenance, application development and customer service. As of now, the infrastructural Cloud computing platform, such as physical servers, storage and network devices, for Guoshi Network is ready and working, deployed over 12 datacenters at Beijing, Wuxi, Dongguan, Tibet, and other cities. Beijing is the first major center, and Wuxi the second, with Tibet acting as a mirroring site, the rest are sub-centers. There are over 1000 servers at these 12 datacenters, and an aggregated computing power of 50Teraflops, and 580TB of storage. Out of the resources, 140 servers and 70TB of storage are assigned to Science@Guoshi for scientific computation and experiments.

The second major center – Wuxi, has 140 servers, each with 8-core processors, and a total storage of 60TB, and 20 servers and 10TB of storage are allocated to Science@Guoshi.

[10] http://science.guoshi.com

We deployed our solution over 3 of the 12 datacenters including Wuxi, Dongguan, and Kaifeng. The deployment diagram is shown in Fig. 14. We chose Wuxi as a demo center on which we ported the Montage application [19], and developed a Nebula Image Mosaic demo service. For the Cloud platform underneath we chose OpenNebula and we used up to 96 VMs.

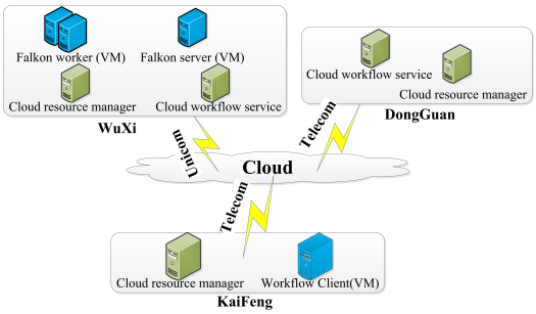

Fig. 14. Deployment on Science@Guoshi

We present the application deployment based on the Montage Image Mosaic Workflow. The Montage Workflow has much larger input size and number of input files (up to tens of thousands), and the workflow can process different nebula image data, which serves better for educational purpose and also demonstrate the scalability of Cloud.

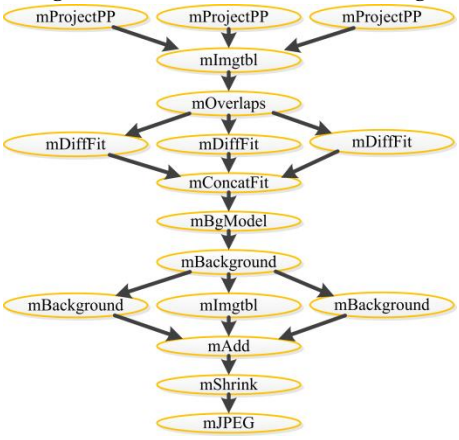

Fig. 15. The Montage Workflow

Montage is a suite of software tools developed to generate large astronomical image mosaics by composing multiple small images, as shown in Fig. 15. The typical workflow process involves the following key steps:

- Image projection:
  - re-project each image into a common coordinate space (mProjectPP)
- Background rectification:
  - Calculate a list of overlapping images (mOverlaps)
  - Perform image difference between each pair of overlapping images (mDiffFit)
  - Fit difference images into a plane (mConcatFit)
  - Background correction (mBackground)
- Image co-addition (mAdd):
  - Optionally divide a region into a grid of sub-regions, and co-add the images in each region into a mosaic
  - Co-add the processed images (or mosaics in sub-regions) into a final mosaic

And finally the mosaic is shrunk (mShrink) and converted into a JPEG image (mJPEG) for display.

In the demo a Science@Guoshi user can pick one of the nebula (as illustrated in Table 4) to create the mosaic for it, and the demo service submits a workflow request to the Cloud workflow service, which in turn instantiates the Cloud resources on-the-fly to execute the workflow. The demo also visualizes workflow progress in a DAG (directed acyclic graph), and displays the execution log and intermediate results. The deployment provides scientists with an easy-to-use platform to manage and execute scientific workflows on a Cloud platform without knowing the details of workflow scheduling and Cloud resource provisioning.

As the Science@Guoshi platform is not yet completely open to the public, so far, there are totally 125 registered users, mainly from 9 research institutes. Besides the Nebula Image Workflow, the scientific computing platform has been utilized by scientists from different institutes to conduct experiments and share their experience. Some application types are shown in Table 5. The platform serves as a gateway to learning and exploring workflow and Cloud technologies for the community.

TABLE 4
NEBULA DATASETS USED IN THE MONTAGE WORKFLOW

Nebula Dataset	# of Workflow Nodes	# of Images	Image Size	Region Size(deg)
Orion	766	168	333MB	1.0
Swan	199	45	90.5MB	0.5
Trifid	235	48	108MB	0.5
Andromeda	244	48	108MB	0.5
Triangulum	235	48	108MB	0.5
Sunflower	322	69	133MB	0.5
Atlas Images	3350	732	1.42GB	N/A

TABLE 5
RESEARCH BASED ON THE SCIENTIFIC COMPUTING PLATFORM

Research	Research Field	Institute
Molecular Simulation	Computational Chemistry	Beijing University of Chemical Technology
Medical Image Analysis	Medical Science	Univ. of Electronic Sci. and Tech. of China
Gene Sequencing	Bioinformatics	Chinese Academy of Sciences
Materials of Metallorganic Frameworks	Material Science	Dalian University of Technology

7 CONCLUSIONS AND FUTURE WORK

As more and more scientific applications are migrating into Cloud, it is imperative to also migrate SWFMSs into Cloud to take advantage of Cloud scalability, and also to handle the ever increasing data scale and analysis complexity of such applications. Cloud offers unprecedented scalability to workflow systems, and could potentially change the way we perceive and conduct scientific experiments. The scale and complexity of the science problems that can be handled can be greatly increased on the Cloud, and the on-demand nature of resource allocation on the Cloud will also help improve resource utilization and user experience.

We propose a reference service framework for integrating scientific workflow management systems into various Cloud platforms, and also present our implementation effort in integrating the Swift workflow management system with the OpenNebula and the Eucalyptus Cloud platforms according to the service framework, in which a client-side tool, a Cloud workflow management service, a Cloud resource manager, and a cluster monitoring service are developed. We also demonstrate the functionality and efficiency of our approach using two real-world scientific workflows.

The implementation can readily be used for OpenStack as it is getting more popularity in scientific research area and commercial applications. We are also investigating the integration of other SWFMSs into these various Clouds.

ACKNOWLEDGMENT

The corresponding authors of this paper are Yong Zhao and Wenhong Tian. This work was supported by the National Science Foundation of China No. 61272528 and No. 61034005, and the Central University Fund (ID-ZYGX2013J073).

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, S. Lu. "Cloud Computing and Grid Computing 360-Degree Compared," IEEE Grid Computing Environments (GCE08) 2008, co-located with IEEE/ACM Supercomputing 2008. Austin, TX. pp. 1-10

[2] Y. Zhao, Y. Zhang, W. Tian, R. Xue, C. Lin, Designing and Deploying a Scientific Computing Cloud Platform. 2012 ACM/IEEE 13th International Conference on Grid Computing, 2012, pp 104-113.

[3] M. Kozlovszky, K. Karóczkai, I. Márton, A. Balasko, A. C. Marosi, and P. Kacsuk, "Enabling Generic Distributed Computing Infrastructure Compatibility for Workflow Management Systems", Computer Science, vol. 13, no. 3, p. 61, 2012.

[4] W. Tan, K. Chard, D. Sulakhe, R. Madduri, I. Foster, S. S.-Reyes, C. Goble, "Scientific Workflows as Services in caGrid: A Taverna and gRAVI Approach," ICWS 2009: 413-420.

[5] G. Papuzzo, G. Spezzano. Autonomic management of workflows on hybrid grid-cloud infrastructure. Proceedings of the 7th International Conference on Network and Services Management. International Federation for Information Processing, 2011: 230-233.

[6] R. Moreno-Vozmediano, R.S. Montero, I.M. Llorente. "Multi-Cloud Deployment of Computing Clusters for Loosely-Coupled MTC Applications", IEEE Transactions on Parallel and Distributed Systems. 22(6), pp.924-930, 2011.

[7] G. Bell, T. Hey, A. Szalay, Beyond the Data Deluge, Science, Vol. 323, no. 5919, pp. 1297-1298, 2009.

[8] E. Deelman et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming, vol. 13, iss. 3, pp. 219-237. July 2005.

[9] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, Concurrency and Computation: Practice and Experience,Special Issue: Workflow in Grid Systems, vol. 18, iss. 10, pp. 1039–1065, 25 August 2006.

[10] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger and H. T. Vo, Managing Rapidly-Evolving Scientific Workflows, Provenance and Annotation of Data, Lecture Notes in Computer Science, 2006, vol. 4145/2006, 10-18, DOI: 10.1007/11890850_2

[11] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," Nucleic Acids Research, vol. 34, pp. 729-732, 2006.

[12] Y. Zhao, X. Fei, I. Raicu, S. Lu, Opportunities and Challenges in Running Scientific Workflows on the Cloud, IEEE International Conference on Cyber-enabled distributed computing and knowledge discovery (CyberC), pp. 455-462, 2011.

[13] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good, "On the Use of Cloud Computing for Scientific Workflows," 3rd International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES), pp. 640-645, 2008.

[14] K. Keahey, T. Freeman, "Science Clouds: Early Experiences in Cloud Computing for Scientific Applications," Cloud Computing and Its Applications 2008 (CCA-08), Chicago, IL. October 2008.

[15] F. Jrad, J. Tao, A. Streit. A broker-based framework for multi-cloud workflows. Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds. ACM, 2013: 61-68.

[16] J. Wang, P. Korambath, I. Altintas, J. Davis, D. Crawl. Workflow as a Service in the Cloud: Architecture and Scheduling Algorithms. Procedia Computer Science, 2014, 29: 546-556.

[17] A. Kashlev, S. Lu, A. Chebotko, "Coercion Approach to the Shimming Problem in Scientific Workflows", in Proc. Of the IEEE International Conference on Services Computing (SCC), Santa Clara , CA, USA, 2013.

[18] E. Apostol, I. Baluta, A. Gorgoi, V. Cristea. Efficient manager for virtualized resource provisioning in cloud systems. Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on. IEEE, 2011: 511-517.

[19] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the Cloud: the Montage example. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, pp. 50:1–50:12, Piscataway, NJ, USA, 2008.

[20] C. Vecchiola, S. Pandey, and R. Buyya. High-Performance Cloud Computing: A View of Scientific Applications. In International Symposium on Parallel Architectures, Algorithms, and Networks, pp. 4-16, 2009.

[21] S. Ostermann, R. Prodan, T. Fahringer. Extending grids with cloud resource management for scientific computing. Grid Computing, 2009 10th IEEE/ACM International Conference on. IEEE, 2009: 42-49.

[22] K. Keahey, T. Freeman. Contextualization: Providing One-click Virtual Clusters. in eScience. 2008, pp. 301-308. Indianapolis, IN, 2008.

[23] G. Juve and E. Deelman. Wrangler: Virtual Cluster Provisioning for the Cloud. In HPDC, pp. 277-278, 2011.

[24] X. Fei, S. Lu: A Dataflow-Based Scientific Workflow Composition Framework. IEEE Transactions on Services Computing (TSC) 5(1):45-58 (2012).

[25] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua, "A Reference Architecture for Scientific Workflow Management Systems and the VIEW SOA Solution," IEEE Transactions on Services Computing (TSC), 2(1), pp.79-92, 2009.

[26] A. Chebotko, S. Lu, S. Chang, F. Fotouhi, P. Yang: Secure Abstraction Views for Scientific Workflow Provenance Querying. IEEE Transactions on Services Computing (TSC) 3(4):322-337 (2010).

[27] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Falkon: a Fast and Light-weight tasK executiON framework," IEEE/ACM SuperComputing 2007, pp. 1-12.

[28] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System, 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, pp. 124-131, 2009.

[29] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, I. Raicu, "Parallel Scripting for Applications at the Petascale and Beyond," IEEE Computer Nov. 2009 Special Issue on Extreme Scale Computing, vol. 42, iss. 11, pp. 50-60, 2009.

[30] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman. Data Sharing Options for Scientific Workflows on Amazon EC2, Proc. ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC'10), 2010.

[31] T. T. Huu, G. Koslovski, F. Anhalt, J. Montagnat, P. V-B Primet. Joint Elastic Cloud and Virtual Network Framework for Application Performance-cost Optimization. Journal of Grid Computing, 2011, Volume 9, Issue 1 , pp 27-47.

[32] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Laszewski, I. Raicu, T. S.-Praun, M. Wilde. "Swift: Fast, Reliable, Loosely Coupled Parallel Computation," IEEE Workshop on Scientific Workflows 2007, pp. 199-206.

Yong Zhao is a professor at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. He obtained his Ph.D. in Computer Science from the University of Chicago under Dr. Ian Foster's supervision. His research areas are in Cloud computing, many-task computing, and data intensive computing. He is a member of ACM, IEEE and CCF.

Youfu Li is a graduate student with the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interest is in Cloud computing, scientific workflows and real-time computing. He is a student member of the IEEE

Ioan Raicu is an assistant professor in the Department of Computer Science at Illinois Institute of Technology. He obtained his Ph.D. in Computer Science from University of Chicago under the guidance of Dr. Ian Foster. He is particularly interested in many-task computing, data intensive computing, Cloud computing, and many-core computing. He is a member of the IEEE and ACM.

Shiyong Lu is an Associate Professor in the Department of Computer Science at Wayne State University and the Director of the Big Data Research Laboratory. He received his PhD in Computer Science from Stony Brook University. His research focuses on big data and scientific workflows. He is a senior member of the IEEE.

Cui Lin is an assistant professor at the Department of Computer Science, California State University. Her research interest is in scientific workflows, services computing and bioinformatics. She is a member of IEEE.

Yanzhe Zhang is a researcher at the Institute of Computing Technology, Chinese Academy of Sciences. His research interest is in high performance computing.

Wenhong Tian is an associate professor at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interest is in resource management and scheduling in Cloud datacenters. He is a member of IEEE.

Ruini Xue is an associate professor at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interest is in distributed computing and Cloud computing. He is a member of IEEE.