2012 Second International Conference on Cloud and Green Computing

# Scientific-Workflow-Management-as-a-Service in the Cloud

Yong Zhao, Youfu Li, Wenhong Tian, Ruini Xue
School of Computer Science and Engineering
University of Electronic Science and Technology of China, Chengdu, China
yongzh04@gmail.com, youfuli.fly@gmail.com, tian_wenhong@uestc.edu.cn, xueruini@gmail.com

*Abstract*— **Scientific workflow management systems have been around for many years and provide essential support such as management of data and task dependencies, job scheduling and execution, provenance tracking, etc. to scientific computing. While we are entering into a "big data" era, it is necessary for scientific workflow systems to integrate with Cloud platforms so as to deal with the ever increasing data scale and analysis complexity. In this paper, we present our experience in offering the Swift scientific workflow management system as a service in the Cloud. Our solution integrates Swift with the OpenNebula Cloud platform, and supports workflow specification and submission, on-demand virtual cluster provisioning, high-throughput task scheduling and execution, and efficient and scalable Cloud resource management. We demonstrate the capability of the solution using a NASA MODIS image processing workflow.**

***Cloud computing; Scientific Workflow; Cloud workflow; Swift; Workflow-as-a-Service***

## I. INTRODUCTION

Scientific workflow management systems (SWFMS) have been proven essential to scientific computing as they provide functionalities such as workflow specification, process coordination, job scheduling and execution, provenance tracking, fault tolerance etc. Systems such as Taverna [11], Kepler [9], Vistrails [10], Pegasus [8], Swift [32], VIEW [26] have seen wide adoption in various disciplines such as physics, astronomy, bioinformatics, neuroscience, earth science, social science etc. Nevertheless, advances in science instrumentation and network technologies are posing new challenges to our trusty workflow systems in both data scale and application complexity.

We are entering into a "big data" era. The amount of data created in the world is growing explosively. According to recent IDC research, the total amount of digital information in the world reached 1 zettabyte in 2010. Popular search engines such as Google and Bing can generate multiple terabytes of search logs every day, social network data is also tremendous: on Facebook, each month the community creates more than 30 billion pieces of content ranging from Web links, news, stories, blog posts and notes to videos and photos [5]. The scientific community is also facing a "data deluge" [7] coming from experiments, simulations, sensors, and satellites. The Large Hadron Collider [4] at CERN can generate more than 100 terabytes of collision data per second; GenBank [3], one of the largest DNA databases, already 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, and can run up to tens of CPU years.

As an emerging computing paradigm, Cloud computing [6] is gaining tremendous momentum in both academia and industry: not long after Amazon opened its Elastic Computing Cloud (EC2) to the public, Google, IBM, and Microsoft all released their Cloud platforms one after another. Meanwhile, several open source Cloud platforms, such as Hadoop [33], OpenNebula [1], Eucalyptus [34], Nimbus [22], and OpenStack [2], become available with fast growth of their own communities.

There are a couple of major benefits and advantages that are driving the widespread adoption of the Cloud computing paradigm: 1) Easy access to resources: resources are offered as services and can be accessed over Internet. For instance, with a credit card, you can get access to Amazon EC2 virtual machines immediately; 2) Scalability on demand: once an application is deployed onto the Cloud, the application can be automatically made scalable by provisioning the resources in the Cloud on demand, and the Cloud takes care of scaling out and in, and load balancing; 3) Better resource utilization: Cloud platforms can coordinate resource utilization according to resource demand of the applications hosted in the Cloud; and 4) Cost saving: Cloud users are charged based on their resource usage in the Cloud, they only pay for what they use, and if their applications get optimized, that will be reflected into a lowered cost immediately.

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 running scientific workflows on top of Cloud can be multifold:

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

978-0-7695-4864-7/12 $26.00 © 2012 IEEE
DOI 10.1109/CGC.2012.70

applications, allowing scientific discoveries to be carried out in an 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 deployed 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 Cloud can improve resource utilization and change the experience of end users for improved responsiveness. Cloud-based workflow applications can get resources allocated accordingly with 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 a fast turn-around time for end users.

4) Cloud computing provides a 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 a 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.

In an early paper [12], we identified various challenges associated with migrating and adapting an SWFMS in the Cloud. In this paper, we present an end-to-end approach that addresses the integration of Swift, an SWFMS that has broad application in Grids and supercomputers, with the OpenNebula Cloud platform. The integration covers all the major aspects involved in workflow management in the Cloud, from client-side workflow submission to the underlying Cloud resource management, thus providing scientific-workflow-management-as-a-service in the Cloud. The rest of the paper is organized as follows: in Section II, we discuss related work in running scientific applications and workflows in Cloud. In Section III, we present our end-to-end integration solution. In Section IV, we demonstrate and analyze our integration using a NASA MODIS image processing workflow, and in Section V, we draw our conclusions and discuss future work.

## II. RELATED WORK

There have been a couple of early explorers that try to evaluate the feasibility, performance, and adaptation of running data intensive and HPC applications on Clouds or hybrid Grid/Cloud environments. Palankar et al. [17] evaluated the feasibility, cost, availability and performance of using Amazon's S3 service to provide storage support to data intensive applications, and also identified a set of additional functionalities that storage services targeting data-intensive science applications should support. Oliveira et al. [35] evaluated the performance of X-Ray Crystallography workflow using SciCumulus middleware with Amazon EC2. These studies provide good source of information about Cloud platform support for science applications. Other studies investigated the execution of real science applications on commercial Clouds [19] [20], mostly being HPC applications, and compared the performance and cost against Grid environments. While such applications indeed can be ported to a Cloud environment, Cloud execution doesn't show significant benefit due to the applications' tightly coupled nature.

There are also endeavors to run workflow applications on top of Clouds. The series of work [13][15] 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 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.

With VGrADS [16], not only the virtual Grid abstraction enabled a more sophisticated and effective scheduling of workflow sets, unifying workflow execution over batch queue systems and cloud computing sites (including Amazon EC2 and Eucalyptus), but also the Virtual Grid Execution System provided a uniform interface for provisioning, querying, and controlling the resources. Its workflow planner could interact with a DAG scheduler, an EC2 planner and fault tolerance sub-components to trade-off various system parameters - performance, reliability and cost.

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 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.

Bresnahan et al. [24] introduced cloudinit.d, a tool for launching, configuring, monitoring, and repairing a set of interdependent VMs in one or a set of IaaS clouds. In addition, as its name suggested, cloudinit.d could launch groups of interdependent VMs and optimize the launch by allowing independent VMs to launch at the same time.

## III. INTEGRATION

In this section, we talk about our end-to-end approach in integrating Swift with the OpenNebula Cloud platform. Before we go into further details of the integration, we will discuss some background information with regard to workflow systems and Cloud integration options.

### *A. Integration Options*

In our early paper [12], we described a reference architecture of SWFMSs, and then identified four integration

approaches to the deployment of SWFMSs in a Cloud computing environment according to the reference architecture. The reference architecture for SWFMSs [25] is proposed as an endeavor to standardize SWFMS research and development efforts, and an SOA-based instantiation is first implemented in the VIEW system. As shown in Figure 1, the reference architecture consists of 4 logical layers, 7 major functional subsystems, and 6 interfaces. The first layer is the Operational Layer, which consists of a wide range of heterogeneous and distributed data sources, software tools, services, and their operational environments, including high-end computing environments. The second layer is called the Task Management Layer. This layer consists of three subsystems: Data Product Management, Provenance Management, and Task Management. The third layer, called the Workflow Management Layer, consists of Workflow Engine and Workflow Monitoring. Finally, the fourth layer – the Presentation Layer, consists of the Workflow Design subsystem and the Presentation and Visualization subsystem. The reference architecture would allow the scientific workflow community to focus on different layers and subsystems of SWFMSs, and also enable such systems to interact and interoperate with each other based on the interface definitions.

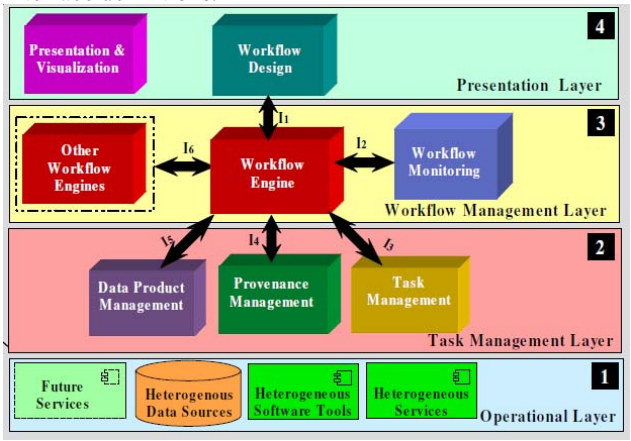


Figure 1 A reference architecture for SWFMSs

The four deployment options, accordingly, correspond to deploying different layers of the reference architecture into 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 the risk associated with vendor lock-in.

### *B. The Swift Workflow Management System*

Swift is a system that bridges scientific workflows with parallel computing. It is a parallel programming tool for rapid and reliable specification, execution, and management of large-scale science and engineering workflows. 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 [31], and dynamic dataset mappings for accessing large-scale datasets represented in diverse data formats. The runtime system provides an efficient workflow engine for scheduling and load balancing, and it can interact with various resource management systems such as PBS and Condor for task execution.

The Swift system architecture consists of four major components: Program Specification, Scheduling, Execution, and Provisioning, as illustrated in Figure 2. Computations are

specified in SwiftScript, which has been shown to be simple yet powerful. SwiftScript programs are compiled into abstract computation plans, which are then scheduled for execution by the workflow engine onto provisioned resources. 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.

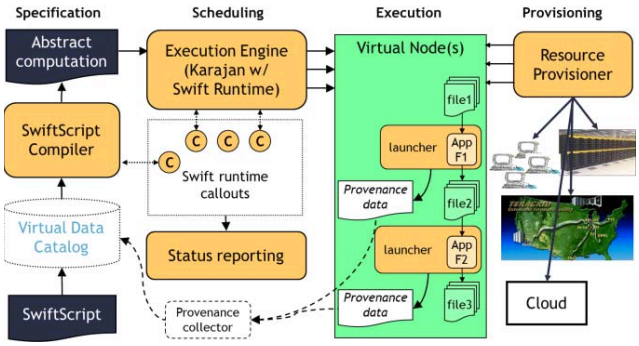


Figure 2 Swift System Architecture

The four major components of the Swift system can be easily mapped into the four layers in the reference architecture: the specification falls into the Presentation Layer, although SwiftScript focuses more on the parallel scripting aspect for user interaction than on Graphical representation; the scheduling components correspond to the Workflow Management Layer; the execution components maps to the Task Management layer; and the provisioning layer can be thought as mostly in the Operational Layer.

### C. *Integration Challenges*

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.

There are a couple of challenges associated with this integration approach. Firstly, we need to port the SWFMS, in our case, Swift into the Cloud, which would usually involve wrapping up an SWFMS as a Cloud service. In addition, to fully explore the capability and scalability of the Cloud, the workflow engine may need to be re-engineered to be able to interact directly with the various Cloud services such as storage, resource allocation, task scheduling, monitoring, etc. At the client side, either a complete Web-based user interface needs to be developed to allow users to specify and interact with the SWFMS, or a thin desktop client application needs to be developed to interact with the SWFMS Cloud service.

Secondly, we need to deal with resource provisioning issue. 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, 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. We take a similar approach that creates a virtual cluster and deploys the Falkon [27] execution services onto the cluster for high-throughput task scheduling and execution. Falkon is a light-weight task execution service for optimized task throughput and resource efficiency delivered by a streamlined dispatcher, a dynamic resource provisioner, and the data diffusion mechanism [28] to cache datasets in local disk or memory and dispatch tasks according to data locality.

### D. *Integration Architecture*

We devise an end-to-end integration approach that addresses the above mentioned challenges. We call it end-to-end because it covers all the major aspects involved in the integration, including a client side workflow submission tool, a Cloud workflow management service that accepts the submissions, a Cloud Resource Manager (CRM) that accepts resource requests from the workflow service and dynamically instantiates a Falkon virtual cluster, and a cluster monitoring service that monitors the health of the acquired Cloud resources.

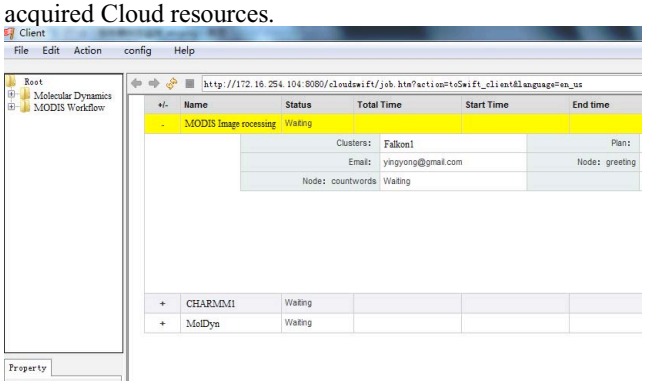


Figure 3 The Client Tool

**The client submission tool:** 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 application 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 in the Cloud. For submission, it provides multiple submission options: execute immediately, execute at a fixed time point, or execute recurrently (per day, per week etc.). We give a

screenshot of the tool in Figure 3, which shows the current status of workflows submitted to the Cloud service.

We integrate Swift with the OpenNebula Cloud platform. We choose OpenNebula for our implementation because it has a flexible architecture and is easy to customize, and also because it provides a set of tools and service interfaces that are handy for the integration. Of course, other Cloud platforms can be integrated in similar means. We show the system diagram of the integration in Figure 4.

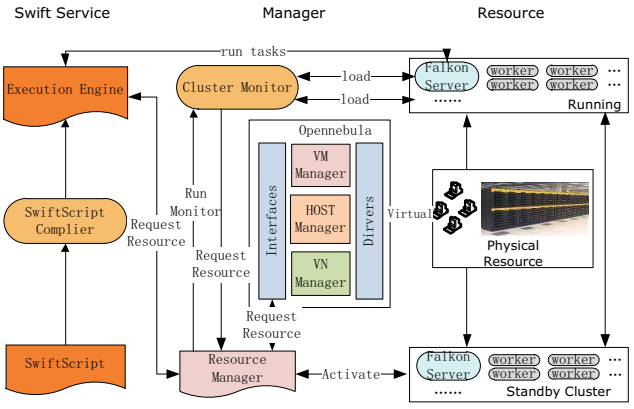


Figure 4 Integration Architecture

**The Swift Cloud Workflow Management Service**

One of the key components of the system is the Swift Cloud workflow management service that 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. A screenshot of the service that visualizes workflow execution progress is shown in Figure 5.

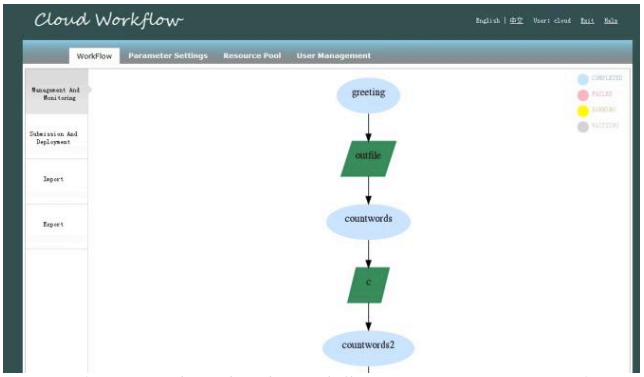


Figure 5 The Cloud Workflow Management Service

**The Cloud Resource Manager**

The Cloud Resource Manager accepts resource requests from the Cloud workflow management service, and is in charge of interfacing with OpenNebula and provisioning Falkon virtual clusters dynamically to the workflow service. In addition, it also monitors the virtual clusters. The process to start a Falkon virtual cluster is as follows:

1) CRM provides a service interface to the workflow service, the latter makes a resource request to CRM.
2) CRM initializes and maintains a pool of virtual machines, the number of virtual machines in the pool can be set via a config file, Ganglia is started on each virtual machine to monitor CPU, memory and IO.
3) Upon a resource request from the workflow service:
   a) CRM fetches a VM from the VM pool and starts the Falkon service in that VM.
   b) CRM fetches another VM and starts the Falkon worker in that VM, and also makes that worker register to the Falkon service.
   c) CRM repeats step b) until all the Falkon workers are started and registered.
   d) If the VMs in the pool are not enough, then CRM will make resource request to the underlying OpenNebula platform to create more VM instances.
4) CRM returns the end point reference of the Falkon server to the workflow service, and the workflow service can now dispatch tasks to the Falkon execution service.
5) CRM starts the Cluster Monitoring Service to monitor the health of the Falkon virtual cluster. The monitoring service checks heartbeat from all the VMs in the virtual cluster, and will restart a VM if it goes down. If the restart fails, then for a Falkon service VM, it will get a new VM and start Falkon service on it, and have all the workers register to the new service. For a Falkon worker VM, it will replace the worker, and also delete the failed VM.
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 CRM 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 VM 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, CRM interacts with and relies on the underlying 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. We

demonstrate the capability and efficiency of our integration using a small scale experiment setup.

## IV. EXPERIMENT

In this section, we demonstrate and analyze our integration approach using a NASA MODIS image processing workflow. The NASA MODIS dataset 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, sand, green land, urban area, etc.

### A. *MODIS Image Processing Workflow*

The workflow (illustrated in Figure 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 area, converts them into displayable format, and assembles them into a single PNG file.

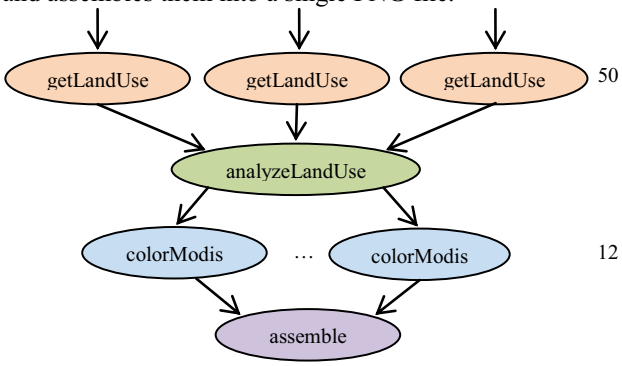


Figure 6 MODIS Image Processing Workflow

### B. *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 operating system is Ubuntu 10.04.1, with OpenNebula 2.2 installed. 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.

### C. *Experiment Results*

In our experiment, we control the workload by changing the number of input data blocks, the resource required, and the submission type (serial submission or parallel submission). So there are three dependent variables. We design the experiment by making two of the dependent variables constant, and changing the other. We run three types of experiments:

1. The serial submission experiment.
2. The parallel submission experiment.
3. The different number of data blocks experiment.

In all the experiments, VMs are pre-instantiated and put in the VM pool. The time to instantiate a VM is around 42 seconds and this doesn't change much for all the VMs created.

1) The serial submission experiment

In the serial submission experiment, we first measure the base line for server creation time, worker creation 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.

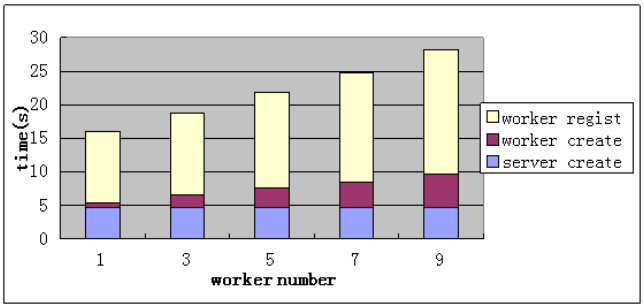


Figure 7 The Base Line for Cluster Creation

In Figure 7, we can observe that the server creation time is quite stable, around 4.7s every time. Worker creation time is also stable, around 0.6s each, and for worker registration, the first one takes about 10s, and for the rest, about 1s each.

For the rest of the serial submission, we submit a workflow after the previous one has finished to test virtual cluster recycling, where the input data blocks remain fixed.

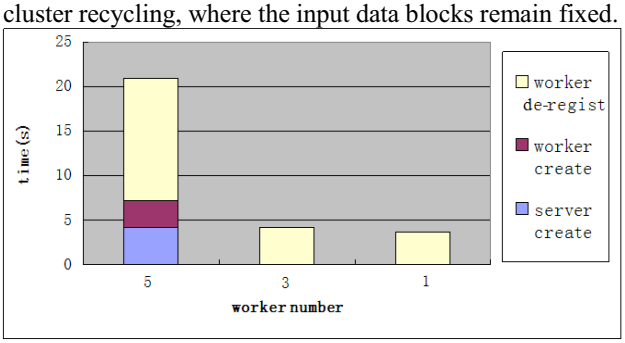


Figure 8 Serial Submission, Decreasing Resource Required

In Figure 8, the resources required are one Falkon server with 5 workers, one server with 3 workers and one server with 1 worker. We can see that for the second and third submissions, the worker creation and server creation time are zero, only the surplus workers need to de-register themselves.

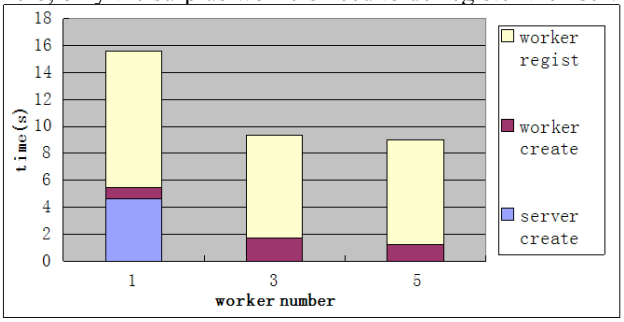


Figure 9 Serial Submission, Increasing Resource Required

In Figure 9, the resources required are in the reverse order of those in Figure 8. Each time two extra Falkon workers need to be created and registered, and the time taken are roughly the same. These experiments show that the Falkon virtual cluster can be re-used after it is being created, and worker resources can be dynamically removed or added.

In Figure 10, we first request a virtual cluster with 1 server and 9 workers, we then make 5 parallel requests for virtual clusters with 1 server and 1 worker. We can observe that one of these requests is satisfied using the existing virtual cluster, where the other 4 are created on-demand. In this case, it takes some time to de-register all the 8 surplus workers, which makes the total time comparable to on-demand creation of the cluster.

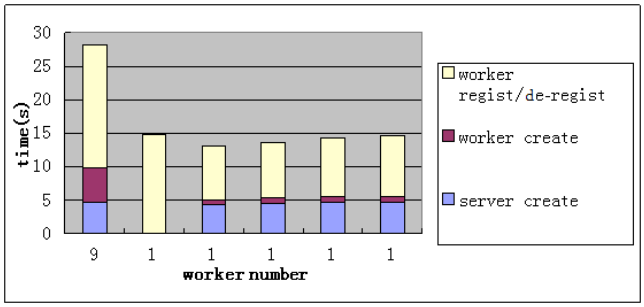


Figure 10 Serial Submission, Mixed Resource Required

2) The parallel submission experiment

In the parallel submission experiment, we submit multiple workflows at the same time in order to measure the maximum parallelism (the number of concurrent workflows that can be hosted in the Cloud platform) in the environment.

| No. of Clusters | Server Init | Worker Creation | Worker Registration |
|---|---|---|---|
| 1 | 4624ms | 1584ms | 11305ms |
| 2 | 4696ms | 2367ms | 11227ms |
| | 445ms | 0 | 0 |
| 3 | 4454ms | 1457ms | 11329ms |
| | 488ms | 0 | 0 |
| | 548ms | 0 | 0 |
| 4 | 521ms | 0 | 0 |
| | 585ms | 0 | 0 |
| | 686ms | 0 | 0 |
| | submission failed | | |

Table 1 Parallel Submission, 1 Server 2 Workers

First, we submit resource requests with 1 server and 2 workers, and the maximum parallelism is up to three. In Table 1, we show the results for the experiment, in which we make resource requests for 1 virtual cluster, 2 virtual clusters, 3 virtual clusters and 4 virtual clusters. For the request of 2 virtual clusters, it can re-use the one released by the early request, and the time to initialize the cluster is significantly less than fresh creation (445ms vs. 4696ms). It has to create the second cluster on-demand. For the 4-virtual-cluster request, since all the VM resources are used up by the first 3 clusters, the 4th cluster creation would fail as expected. When we change resource requests to 1 server and 4 workers, the maximum parallelism is two, and the request to create a third virtual cluster also fails. Since our VM pool has a maximum of ten virtual machines, it's easy to explain why this has happened. This experiment shows that our integrated system can maximize the cluster resources assigned to workflows to achieve efficient utilization of resources.

3) 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 total execution time with varying number of workers in the virtual cluster.


Figure 11 Different Input Sizes

In Figure 11, 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 can't subsidize the management and registration overhead of the added worker. The time for server and worker creation, and worker registration remain unchanged when we change the input size (as have been shown in Figure 7). The experiment indicates that while our virtual resource provisioning overhead is well controlled, we do need to carefully determine the number of workers used in the virtual cluster to achieve resource utilization efficiency.

## V. CONCLUSIONS

As more and more scientific applications are migrating into Cloud, it is important to also migrate SWFMSs into Cloud to take advantage of Cloud scalability, and 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 present our early effort in offering workflow management as a service by integrating the Swift workflow management system with the OpenNebula Cloud platform, in which a Cloud workflow management service, a Cloud resource manager, and a cluster monitoring service are developed. We also conduct a set of experiments to showcase the functionality and efficiency of our approach. For future work, we will also leverage distributed storage for VM images, and conduct

large scale experiments to look at ways to improve VM instantiation, virtual cluster creation and workflow execution. We are also investigating the integration of Swift with the Openstack Cloud platform, as Openstack is gaining popularity in the science community.

## ACKNOWLEGEMENT

This paper is supported by the key project of National Science Foundation of China No. 61034005 and No. 61073175.

## REFERENCES

[1] OpenNebula, http://www.OpenNebula.org, 2012

[2] Openstack, http://www.openstack.org, 2012

[3] http://www.psc.edu/general/software/packages/genbank/,2012

[4] Large Hadron Collider, http://lhc.web.cern.ch, 2012

[5] Shawn Rogers, Big Data is Scaling BI and Analytics, Information Management, Sept 1, 2011.

[6] 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.

[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, Volume 13 Issue 3, 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, Volume 18, Issue 10, pages 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, Volume 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, iss. Web Server issue, 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), 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), Indianapolis, Indiana, USA, December 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] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, G. B. Berriman. "Experiences Using Cloud Computing for A Scientific Workflow Application", Invited Paper, ACM Workshop on Scientific Cloud Computing (ScienceCloud) 2011.

[16] L. Ramakrishnan, C. Koelbel, Y.-S. Kee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, T.M. Huang, K. Thyagaraja, and D. Zagorodnov, "VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance," Proc. Conf. High Performance Computing Networking, Storage and Analysis (SC'09), 2009.

[17] M. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel. Amazon S3 for science grids: a viable solution? In Proceedings of the 2008 international workshop on Data-aware distributed computing (DADC '08). 2008.

[18] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, D. H. Epema, "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing," IEEE Transactions on Parallel and Distributed Systems, pp. 931-945, June, 2011.

[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, pages 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 onParallel Architectures, Algorithms, and Networks,2009.

[21] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon et al. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In CloudCom, pages 159–168. IEEE, 2010.

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

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

[24] Bresnahan, J., Freeman, T., LaBissoniere, D., Keahey, K., Managing Appliance Launches in Infrastructure Clouds, Teragrid 2011. Salt Lake City, UT. July 2011.

[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] C. Lin, S. Lu, Z. Lai, A. Chebotko, X. Fei, J. Hua, F. Fotouhi, *"Service-Oriented Architecture for VIEW: a Visual Scientific Workflow Management System",* In Proc. of the IEEE 2008 International Conference on Services Computing (SCC), Honolulu, Hawaii, USA, July 2008, pp.335-342.

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

[28] I. Raicu, Y. Zhao, I. Foster, A. Szalay. "*Accelerating Large-scale Data Exploration through Data Diffusion*", International Workshop on Data-Aware Distributed Computing 2008, co-locate with ACM/IEEE International Symposium High Performance Distributed Computing (HPDC) 2008.

[29] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, Allan Espinosa, Mihael Hategan, Ben Clifford, Ioan Raicu. "*Parallel Scripting for Applications at the Petascale and Beyond*", IEEE Computer Nov. 2009 Special Issue on Extreme Scale Computing, 2009.

[30] NASA MODIS dataset, http://modis.gsfc.nasa.gov/, 2012.

[31] Y. Zhao, J. Dobson, I. Foster, L. Moreau, M. Wilde, *A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data,* SIGMOD Record, Volume 34, Number 3, September 2005.

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

[33] Hadoop, http://hadoop.apache.org/, 2012

[34] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The Eucalyptus Open-Source Cloud-Computing Platform, In Proceedings of Cloud Computing and Its Applications 2008.

[35] Oliveira, D. Ocaña, K., Ogasawara, E., Dias, J., Baião, F., Mattoso, M.: A Performance Evaluation of X-Ray Crystallography Scientific Workflow Using SciCumulus. IEEE CLOUD 2011, p. 708-715