A System Architecture for Running Big Data Workflows in the Cloud

Andrey Kashlev, Shiyong Lu
Department of Computer Science
Wayne State University

Abstract—Scientific workflows have become an important paradigm for domain scientists to formalize and structure complex data-intensive scientific processes. The ever-increasing volumes of scientific data motivate researchers to extend scientific workflow management systems (SWFMSs) to utilize the power of Cloud computing to perform big data analyses. Unlike workflows run in traditional on-premise environments such as stand-alone workstations or grids, Cloud workflows rely on dynamically provisioned computing, storage and network resources that are terminated when no longer used. This dynamic and volatile nature of cloud resources as well as other cloud-specific factors introduce a new set of challenges for “Cloud-enabled” SWFMSs. Although several SWFMSs have been developed to use cloud resources, they are geared towards either a specific domain such as bioinformatics [4, 5] or astronomy [6], or a particular type of workflows such as workflows with parameter sweep and data fragmentation parallelism [7], or workflows with time and cost QoS constraints for each task [8]. While such ad hoc implementations account for particularities of their target applications, they do not address the breadth of challenges in managing scientific workflows in the cloud. Thus, more generic, implementation-independent solution is needed that would address a broader scope of cloud-related challenges in a systematic way. This can be achieved through a comprehensive study of cloud-related challenges from an architectural perspective. To address this need, in this paper we:

1. identify the key challenges of running big data workflows in the cloud,
2. propose a generic implementation-independent system architecture that addresses these challenges,
3. develop a cloud-enabled SWFMS called DATAVIEW that delivers a specific implementation of the proposed architecture. Finally, to validate our proposed architecture we conduct a case study in which we design and run a big data workflow towards addressing EB-scale big data analysis problem in the automotive industry domain.

Keywords-component; scientific workflow; cloud; big data;

I. INTRODUCTION

Scientific workflows are widely recognized to be an important paradigm in the services computing field [1, 2] as they allow data scientists to compose together various heterogeneous services into data processing pipelines that facilitate research and discovery in various domains of e-science. A scientific workflow management system (SWFMS) is a software system that allows domain scientists to design, store and execute scientific workflows to solve their domain problems. As scientists need to process data of high volume, velocity, and variety, it is imperative to enable SWFMSs to use distributed computing and storage resources available in the cloud to run these so called big data workflows [3]. Doing so would (1) facilitate better resource utilization as cloud services are only used on as-needed basis, (2) help domain scientists to achieve better tradeoff between performance and cost of data analyses, and (3) enable scalability on demand when more data need to be processed by a workflow than originally expected or when more scientists need to use a collaborative SWFMS concurrently.

Unlike workflows run in traditional on-premise environments such as stand-alone workstations or grids, Cloud workflows rely on dynamically provisioned computing, storage, and network resources that are terminated when no longer needed. This dynamic and volatile nature of cloud resources as well as other cloud-specific factors introduce a new set of challenges for “cloud-enabled” SWFMSs.

Although several SWFMSs have been developed to use cloud resources, they are geared towards either a specific domain such as bioinformatics [4, 5] or astronomy [6], or a particular type of workflows such as workflows with parameter sweep and data fragmentation parallelism [7], or workflows with time and cost QoS constraints for each task [8]. While such ad hoc implementations account for particularities of their target applications, they do not address the breadth of challenges in managing scientific workflows in the cloud. Thus, more generic, implementation-independent solution is needed that would address a broader scope of cloud-related challenges in a systematic way. This can be achieved through a comprehensive study of cloud-related challenges from an architectural perspective. To address this need, in this paper we:

1. identify the key challenges of running big data workflows in the cloud,
2. propose a generic implementation-independent system architecture that addresses these challenges,
3. develop a cloud-enabled SWFMS called DATAVIEW that delivers a specific implementation of our architecture and present a case study from the automotive industry domain to validate our solution.

We have tested our DATAVIEW implementation of the proposed architecture in three different cloud environments – Amazon EC2, FutureGrid Eucalyptus and FutureGrid OpenStack IaaS clouds. We ran a workflow analyzing 3 Gb of vehicle data in OpenXC format [14]. As the average adult driver in the US may generate up to 75 Gb of such driving data annually, the total amount of data generated in the US may exceed 14 Eb (10^{18} bytes) per year [22, 23].

II. MAIN CHALLENGES FOR RUNNING SCIENTIFIC WORKFLOWS IN THE CLOUD

Scientific workflows can be thought of as data pipelines consisting of heterogeneous software components connected to one another and to some input data products [2, 9]. These components may include local executable programs, scripts, Web services, HPC jobs, etc. Such workflows are designed using scientific workflow management systems, which
provide domain scientists with intuitive, user-friendly interfaces to design and execute data intensive workflows. SWFMSs will help remove technical burdens from researchers, allowing them to focus on solving their domain-specific problems.

While cloud computing opens many exciting opportunities for running scientific workflows, it also poses several challenges that are not present when running workflows in traditional on-premise environments. As we explain below, several aspects of cloud computing make it more difficult to maintain usability and user-friendliness of SWFMSs. In our work we run the entire system in the cloud, according to the “all-in-the-cloud” approach [9]. The system is deployed on a virtual machine in the cloud (master node) and is accessed remotely through a Web-based GUI interface. SWFMS schedules workflows to run on multiple virtual machines (slave nodes) such that different parts of workflows run on different nodes. We now describe major cloud-related challenges and their impact on scientific workflow management in the cloud.

A. Platforms Heterogeneity Challenge

As cloud computing is still a relatively young field, there is no single universally accepted standard for communicating with the cloud, provisioning resources and managing virtual machine images. Heterogeneity of existing cloud platforms hinders workflow management in the cloud at several levels.

1) Connecting to the cloud

The process of connecting to a particular cloud is defined by the cloud provider and is generally different for different vendors. Connecting to the cloud typically involves providing a security key, and in some cases performing initial configuration (e.g., sourcing euca2ools and novarc), and loading client software (e.g., euca2ools and novacient), as in the case of both Eucalyptus and Openstack clouds [10]. On the other hand, consider the process of accessing a remote server via ssh. Since ssh is an established standard, connecting to any new server is a well-defined procedure requiring no learning effort from users. However, connecting to a cloud is technically more challenging as this process varies by vendors, which puts a burden on the user of having to learn multiple vendor-specific connection protocols and APIs.

2) Resource provisioning

The interfaces exposed by various providers to provision cloud resources are also different (although in some cases slightly different). There exists no standard for provisioning resources in different clouds in a uniform way. For example, while Amazon EC2 [11] provides Java API tools to manage its cloud resources programmatically, OpenStack [12] provides RESTful interface as well as python and command line implementations of OpenStack Nova API.

3) Creating machine images

Bundling, uploading, and registering images also vary by different cloud platforms. In Eucalyptus, an image of a running instance is created by executing the euca-bundle-vol command inside the instance, which produces and saves the image file within the file system of that instance. Because it uses local drive of the VM, it requires large amount (e.g., 6 Gb) of free disk space which may not be available and may be difficult to arrange. In Openstack, on the other hand, the nova image-create command is run to save image file outside of virtual machine (VM) whose state is captured by the snapshot.

4) Migrating workflows between cloud platforms

Oftentimes after running a workflow in one cloud, the user may want to switch to another cloud (e.g., for a better price or customer service). Choosing the number and types of instances to be provisioned in the target cloud environment is a critical step as it determines how long the workflow will run, and the cost of execution if the cloud is proprietary. This is particularly relevant to big data workflows that can run many hours or days. However, various cloud providers support different sets of instance types. For example, Amazon EC2 offers twenty seven instance types, while Openstack offers six types. The types of instances the user had employed in the original cloud may not be supported by the target cloud. Indeed, there is no equivalent of OpenStack’s m1.tiny instance type in Amazon EC2. Thus it is often non-trivial to allocate an equivalent set of machines in the target cloud.

Therefore, such platform heterogeneity makes it challenging to access clouds of different vendors and provision virtual resources in a uniform way. Besides, inconsistent instance types complicate migration from one cloud to another.

B. Resource Selection Challenge

Running a workflow in the cloud always involves the choice of the number and types of virtual machines to execute the workflow. Given a particular configuration (e.g., four m1.large, seven m3.large, and three c1.medium servers), it is hard to determine an optimal schedule, and hence an optimal running time, since the scheduling problem is NP-complete in general. Thus, it is challenging to compare which configuration is better and to choose the best configuration for a given workflow, especially given the exponential size of the search space.

Consider a sample workflow shown in Fig. 1. If the user chooses to run this workflow in Amazon EC2 cloud using three virtual servers, there are $2^3 = 19,683$ possible choices for instance types for the three servers, since EC2 offers 27 instance types. This number will grow exponentially if the user would like to employ more VMs (e.g., for workflows with larger degrees of parallelism).

C. Resource Volatility Challenge

Cloud computing allows to provision and terminate virtual servers and storage volumes on demand. However, due to various failures, loss of resources often occurs (e.g., VMs
crashed). Such dynamic nature of cloud resources has several important implications on scientific workflow management in the cloud as we explain in the following.

1) Persisting output data products
As the workflow execution occurs in the cloud, the output data products that are of interest to the users are also initially saved in the cloud. After execution is complete, user may often need to terminate the instances on which it was running, to avoid paying for the unused virtual servers. Thus, the SWFMS should provide a way to persist output data products to avoid their loss upon terminating virtual machines. This task may be non-trivial in the case of big data workflows with large output files. The user may want to have the option of saving files on his system (client PC) or to place them in a reliable storage, such as Amazon S3. In some cases, users may want to download only output files whose size is under certain threshold (e.g., if the file is 1 GB or less, download it to the client machine, otherwise – store it in S3 bucket).

2) Registering new components or data products
In the dynamic and collaborative environment, users often share their work with each other, oftentimes in the form of scripts or Web services. These new components can be registered with the SWFMS and used for composing new workflows. While on a single machine addition of a new component is only performed once, for a SWFMS running in a virtual machine in the cloud a one-time registration of a component is not sufficient since upon machine termination this update will be lost. The same applies to new data products added to a virtual machine. For example, the user may want to add new interesting datasets to use in future workflows. However, unless precautions are taken, these files may be lost upon terminating the VM.

3) Cataloging virtual resources
Running workflows in the cloud involves executing individual components, residing in different virtual machines, which requires connection-related details for each VM, such as its IP address, credentials (username, password, public key), and status information. It is a challenge to capture in a timely manner changes in VM configurations, their status information, and other metadata. For example it is hard to capture the moment when VM becomes available for use, since cloud providers often prematurely report that the machine is “available”.

Additional challenges may arise when VM’s are accessed for the first time using ssh, requesting to add their public key to the known_hosts file of the client. Thus, although the instance is running, it may not be ready for use in workflow execution – the situation that can prevent workflow from running. Our experience with running scientific workflows in the cloud environment shows that, if overlooked, such seemingly insignificant nuances lead to numerous workflow failures. Similar cataloging should be done for any other virtual resources (e.g. S3 buckets with output data products, machine images, etc.)

4) Environment setup
Scientific workflows are often built from components requiring certain libraries and packages to run. As we explain in further sections, the ComputeGrade component from sample workflow in Fig. 1 relies on the Apache Mahout software to classify a driver’s profile. Running the ComputeGrade component in the cloud requires a virtual machine with Apache Mahout installed on it. However, even if one creates a VM instance and manually installs Mahout on it, once workflow execution finishes and the machine is terminated, re-running the workflow requires provisioning another virtual machine and installing Apache Mahout again. Other components may have entirely different sets of dependencies. While on a single node machine, resolving these dependencies is a one-time procedure, in the cloud environment such configuration would be lost upon terminating the virtual machine. Thus, it is a challenge to provision a set of virtual machines each of which satisfies all dependencies of workflow component(s) scheduled to run on it.

In summary, the volatile nature of cloud resources imposes a challenge of persisting output files and newly registered workflow components and data products in case if all VM’s are terminated. It is also a challenge to keep track of dynamically changing list of virtual machines and credentials to each virtual server and to track which of these machines is ready to run workflows. Finally, creating VM’s suitable to execute workflow components is a challenge, given unique dependencies of each component.

D. Distributed Computing Challenge
The fact that the workflow execution is performed in a distributed manner complicates big data workflow management in several ways.

1) Passing big data products to consumer components
Unlike a single-machine workflow run, cloud-based workflow execution involves components that consume data that physically reside in other virtual machines. Supplying all data products required by a particular component requires knowing hostnames or IP addresses of each VM storing these data products. This in turn requires keeping
track of where every data product resides. The latter can be a non-trivial task in case of large number of dynamically created/deleted VM and data products. Besides, as virtual networks in the cloud environments are normally slower than physical networks used in other infrastructures such as grid or cluster, it is a challenge to efficiently move large data from upstream components to downstream components, especially given the size of big data products.

2) Logging & workflow monitoring

The fact that execution occurs in multiple machines complicates logging process, especially if the cloud network bandwidth is limited. Even sending a simple one-word status update message from one node to another during workflow execution message may incur tangible delay. Therefore, it is challenging to log workflow execution in a distributed environment without slowing down workflow execution. Same challenges apply to monitoring the statuses of individual workflow components.

3) Provenance Collection

Since different components generally execute inside different virtual machines, collecting the data derivation history and storing it on the master node is a challenge.

III. A SYSTEM ARCHITECTURE FOR BIG DATA-ORIENTED SWFMS IN THE CLOUD

We now present our proposed SWFMS architecture, implemented in the DATAVIEW system, shown in Fig. 2.

The main subsystems of DATAVIEW are Workflow Design and Configuration, Workflow Presentation and Visualization, Workflow Engine, Workflow Monitoring, Data Product Management, Provenance Management, Task Management, and Cloud Resource Management. The Presentation Layer contains the client-side part of the system. The Workflow Management Layer contains subsystems orchestrating the progress of the data flow. The Task Management Layer contains modules that ensure successful execution of individual tasks in the cloud. Finally, the Infrastructure Layer contains the underlying IaaS cloud platforms where workflows are dispatched. According to the “all-in-the-cloud” approach [9], the DATAVIEW system runs in the master node (see Fig. 2d). The modules of DATAVIEW that are necessary to run a portion of the workflow on a single machine (but not to coordinate distributed workflow execution) are called DATAVIEW Kernel, which is deployed on each of the slave nodes created at runtime. The master node is responsible for all the “housekeeping” work and coordinating associated with workflow execution and storage. It is not intended to perform actual data processing during the workflow run and thus it does not require high performance virtual machine, which reduces the cost of workflow management in the cloud. We now present an overview of each of the subsystems of DATAVIEW.

The Workflow Design & Configuration subsystem provides intuitive GUI for users to design workflows as well as specify workflow configuration. It consists of two major components. Design component provides a web-based GUI allowing users to compose, edit and save workflows. Workflows are edited in the browser window (see Fig. 1) by dragging and dropping components and input data products onto the design panel and connecting them to the workflow. Once workflow is composed and saved, the scientist uses the Configuration component, which allows users to define the cloud-related workflow settings using a dialog window. First, the user selects among the available cloud providers (e.g., AWS, FutureGrid, Rackspace, etc.). Then he chooses the number of nodes and an instance type for each node. To help the user make the decision, the system dynamically updates the estimated running time of the workflow as well as estimated cost given the current configuration. Once resources are chosen, the user presses the “Run workflow” button which sends a request to the Workflow Engine to run the workflow. The latter forwards provisioning-related information to the Cloud Resource Manager that provisions virtual machines (slave nodes) according to the user’s request. Once requested VMs have been provisioned, the Workflow Engine executes the workflow. This user-friendly interface addresses several challenges outlined earlier – A.1, A.2, and B. The system contains the functionality to connect to different clouds, provision and select resources thereby freeing the user from having to do it manually.

The Workflow Engine is a central subsystem enabling workflow execution. Its architecture is shown in Fig. 2b.

The Translator module is responsible for producing executable representations of workflows (in the case of DATAVIEW these are Java objects) from the specifications written in our XML-based SWL language (Scientific Workflow Language). These specifications are stored in the Workflow Specification Repository. Workflow Configuration Management module captures required cloud-related settings to run the workflow. These include the type of scheduler being used (HEFT, CPOP, etc.), number and types of nodes in the cloud, and mapping of each component to the node where it is scheduled to execute. As these settings are specific to each workflow and even to each workflow run and thus are dynamically changed, they are stored in memory. Dataflow management moves data products within a virtual machine to ensure that every component receives each of its input data products as soon as it is produced by an upstream component. Once all input data are available, the component executes. After component execution is finished, its output data is passed to component-consumers (downstream components) and those of them that are ready (i.e. all input data products are available) are executed. The process continues until all components execute, or until there are no components that are ready to execute. The latter occurs when, say one of the components fails. The EBS Volume Management module leverages Elastic Block Storage volumes to reduce workflow running time. EBS volumes [11] are raw block devices that can be attached to running VM instances. For example, consider a sample workflow scheduled to run in
the cloud using three virtual machines in Fig. 2d VM₁, VM₂, and VM₃, as shown in Fig. 1. Suppose the AnalyzeGasBrk component produced a large output file on VM₁ that needs to be moved to the VM₂ where ComposeProfile is scheduled to execute. Instead of sending a large file over the network, the system attaches an EBS volume to VM₁, stores output of AnalyzeGasBrk on that volume, detaches the volume from VM₁, and attaches the volume to VM₂, avoiding copying the file over the network altogether. Thus, the EBS Volume Management addresses the D.1 challenge (supplying big data products to consumer components).

The Profile Tracker module captures execution times of each component as well as the corresponding runtime performance context during workflow run. The runtime performance context describes factors affecting component’s running time, such as the size and file type of each input data product, the instance type of virtual machine where component is running (e.g., m₃.xlarge, c₂.xlarge, etc.), and the usage of CPU and memory by this component. This information is persisted in Runtime Performance Logs Storage. This addresses the D.2 challenge (logging & workflow monitoring).

When the user attempts to schedule a workflow, the Runtime Behavior Analytics module uses runtime performance context of each workflow component to predict its running time and the overall workflow running time and cost, for the run configuration selected by the user (i.e. the number and types of virtual servers). Runtime Behavior Analytics also enables guided semi-automated cloud resource selection by generating hints suggesting possible improvements the user can make to reduce running time. For example, if certain component is CPU-intensive, the system may suggest using compute optimized instances such as c₂.xlarge, over the general purpose m₃.xlarge to improve performance. Runtime Behavior Analytics relies on profile information collected previously to make such predictions and generate hints. Due to the nature of big data workflows decisions on the number and types of instances are of great importance as they dramatically affect workflow running time. Our semi-automated scheduling process partially addresses the resource selection challenge (B).

The Provenance Collector captures data derivation history in appropriate format such as OPMO [13] and sends it to the Provenance Manager to be stored. This addresses the provenance collection challenge (D.3).

The Workflow Monitoring subsystem keeps track of the statuses of individual components such as “initialized”, “executing”, “finished”, “error”. Oftentimes, one or several of the intermediate components of the workflow may fail and workflow re-run is needed. To save time, it is helpful to “pick up” workflow execution from where it was left after the partially successful run. Keeping track of which components have successfully finished and produced output data enables such smart re-runs. The monitoring
information is sent from each component to the master node. Besides smart re-run, workflow monitoring is crucial as it enables profiling (capturing component performance information), logging and debugging. Thus, the Workflow Monitoring subsystem addresses the D.2 challenge.

The Data Product Management subsystem stores all data products used in workflows. Initially, all data products reside on the master node. Those data products that are used by slave nodes are sent to the corresponding VMs before the workflow execution begins. This addresses the D.1 challenge.

The Provenance Management subsystem is responsible for storing, browsing, and querying workflow provenance.

The Task Management subsystem enables executing heterogeneous atomic tasks such as Web services and scripts.

The Cloud Resource Management (CRM) subsystem plays a key role in provisioning, cataloging, configuring, and terminating virtual resources in the cloud. Its architecture is shown in Fig. 2c.

The CRM subsystem consists of seven modules. The VM Provisioning module is responsible for creating virtual machines from images saved beforehand. These images include the DATAVIEW Kernel needed to run workflows. Machine Image Management maintains a catalogue of machine images (e.g., Amazon and Eucalyptus Machine Images, or AMIs and EMIs respectively) and metadata for each image. These metadata along with all other metadata about available virtual resources are stored in Virtual Resources Catalogue, which addresses the C.3 challenge (cataloging virtual resources). The machine image metadata include operating system, cloud provider, cloud platform, dependencies satisfied in the image, libraries and software installed, etc. The system relies on these metadata and on the schedule to determine which machine image to use when provisioning a VM to run a particular component. For example, when provisioning VM for the ComputeGrade component, the system will choose an image containing the Apache Mahout – a required software to compute the driver’s grade. In this way the system ensures that the provisioned virtual machines have correct execution environment to run workflow components, which addresses challenge C.4 (environment setup).

The EBS Volume Provisioning module creates block storage volumes used by the EBS Volume Management module of the Workflow Engine to efficiently move big data in the cloud, which addresses challenge D.1. Once an EBS volume is created and attached to the running instance, it generally requires formatting, an operation that can take up to several minutes. To avoid such a delay, DATAVIEW relies on snapshots that already contain file system to create EBS Volumes. For this purpose, CRM contains the Snapshot Management module that maintains a list of volume snapshots in the Virtual Resource Catalogue. Snapshot Management is responsible for updating the list and for communicating to the EBS Volume Provisioning module which snapshot is needed for a particular workflow.

S3 Provisioning persists output data products to ensure that after slave nodes are terminated, the data is still available. This addresses challenge C.1 (persisting output data products).

VM Access Management module captures information required for accessing virtual machines, such as credentials, security keys, paths to the DATAVIEW system folders, environment variable names, etc.

IV. IMPLEMENTATION AND CASE STUDY

We implemented the proposed DATAVIEW architecture as a Web-based application, written in Java. To test our implementation and validate our proposed architecture we have conducted a case study by deploying DATAVIEW in Amazon EC2 [11] as well as the Futuregrid’s Eucalyptus and Openstack [10] and running a big data workflow from the automotive domain. As the results were similar in different cloud environments, due to space limit here we report the results obtained in the Amazon EC2. The implementation and case study show how our architecture addresses the challenges A.1-A.3, C.2-C.4, D.1, D.2. We are extending system functionality to address other challenges.

A. DATAVIEW Implementation in the Cloud

DATAVIEW is based on our earlier general purpose SWFMS called VIEW (www.viewsystem.org). It extends VIEW with additional functionality that enables workflow execution in the cloud environment, including the new Workflow Engine and Cloud Resource Manager subsystems. Our CRM subsystem programmatically provisions, configures, and terminates virtual resources (in the case of EC2 using AWS SDK for Java). To create slave nodes, we have registered in the cloud several VM images with DATAVIEW Kernel.

B. Case Study: Analyzing Driving Competency from the Vehicle Data.

We have built a big data workflow analyzing driver’s competency on the road. Our workflow, (Fig. 1) takes as input dataset in the OpenXC format [14]. OpenXC is a platform that allows to collect vehicle data while on the road, using a hardware module installed in the car. The collected data includes steering wheel angle, vehicle speed, accelerator pedal position, brake pedal status, etc. For our experiments we have created a synthetic dataset built from the real data recorded while driving in Manhattan, NY [14].
Our dataset is equivalent to 1 hour worth of data, collected from 50 drivers making the size of the input file 3Gb [23]. The workflow derives competency of each driver based on: 1) How often does the driver accelerate and then suddenly brakes? (AnalyzeGasBrk) 2) How smoothly does the driver accelerate? (AnalyzeSpeedup) and 3) How gradually does the driver brake before making a turn? (AnalyzeBrkngTurns)

Our workflow first extracts data related to acceleration and braking, speedup, and braking before turns using ExtrGasBrk, ExtrSpeedup, and ExtrBrkngTurns components. Then it analyzes each of these three factors and derives a number characterizing each of the three aspects of driving. The lower the number is the better the driver is at this aspect. Once these three numbers are obtained for each driver, they are composed into a driving profile (csv file) by the ComposeProfile component. This profile is then passed to a ComputeGrade component, which uses a classifier called driver.model, built as a logistics regression using Apache Mahout. The ComputeGrade module uses the classifier to determine whether the driver has passed the competency test and produces a final result of the workflow – driving skill assessment report, which is displayed in a pop-up window by DATAVIEW (Fig. 3).

Although the version of statistical analysis algorithms used in this study is relatively simple, we are currently improving its accuracy to account for the fine nuances of the vehicle driving and developing more sophisticated algorithms to assess the driving skill. For the purpose of experiments and to better test our DATAVIEW architecture in the cloud we have injected a dummy CPU-intensive code into the AnalyzeGasBrk, AnalyzeSpeedup, and AnalyzeBrkngTurns components.

In Fig. 4a we report the performance study results from running our scientific workflow in the Amazon EC2. Our system used the HEFT algorithm [15] to schedule the workflow onto the VMs. As shown in Fig. 4a, workflow analysis time decreases when more slave nodes involved in running the workflow as more machines are used to perform the same amount of data processing. As we explain in the next subsection, we ran the workflow in two modes: 1) moving the data to target virtual machines using traditional file transfer protocol scp, and 2) moving the data using the proposed EBS volume movement technique. In the first case the total workflow running time was 8,569, 6676, and 4253 seconds for one, two, and three slave nodes respectively. When using our proposed technique the makespan decreased to 8391, 6047, and 3283 sec. for one, two, and three nodes respectively. Faster data movement technique reduced the makespan in all three configurations. The time to provision VMs averaged at 27 seconds.

C. Moving Big Data within the Cloud.

We have implemented our proposed big data movement technique that supplies large files to target VMs by attaching EBS volumes containing required files to the virtual machines that consume these files. To test our technique we have measured the time to transfer our 3 Gb dataset from one virtual machine to another when using traditional file transfer protocol and when using our proposed technique. The results are shown in Fig. 4b.

As the obtained results confirm, the proposed technique allows to transfer big data files at reasonable rates even when network performance is limited. We assume that the EBS volume used to supply data to the target virtual machine exists in the same region as the machine itself. Since the region of the volume is specified explicitly at volume creation time and thus is in our control, this assumption is easy to meet. The higher the fraction of data movement time is in the overall execution time, the larger is the performance gain attained with our EBS volume movement technique. This explains why the performance improvement is higher for three nodes than for two or one node (Fig. 4a), since more nodes require more data movement. For more data-intensive workflows such performance gain is even larger.

V. RELATED WORK

The need to utilize cloud computing to run scientific workflows has been widely recognized by the scientific community [9]. Many researchers studied and confirmed the feasibility of using cloud computing for e-science from both cost [16] and performance perspectives [17].

In [18], E. Deelman describes mapping workflows onto grid resources, discusses various techniques for improving performance and reliability, and reflects on their use in the
cloud. Zhao et al. discuss various challenges for running scientific workflows in the Cloud as well as identify research directions in this area [9].

In [6], Vöckler et al. demonstrate that the Condor system and the DAGMan engine, originally developed for running jobs in the grid environment can also be extended to run workflows in the cloud. In [8], Wu et al. focus on QoS-constraint based scheduling of workflows in clouds. The authors discuss at high level the architecture of their system running in a simulated cloud. Oliveira et al. [7] present SciCumulus, a cloud middleware that explores parameter sweep and data fragmentation parallelism in scientific workflow activities. The authors present a conceptual architecture and run their system in the simulated cloud. In [4] Abouelhoda et al. propose a system called Taverna which allows seamless integration of the Taverna system with Galaxy workflows based on hierarchical workflows and workflow patterns. Taverna has an interface to set up a cluster in AWS cloud and use it to run workflows. Wang et al. [19] report preliminary work and experiences of enabling the interaction between Kepler SWFMS and the EC2 cloud.

While these solutions provide some insights into development of SWFMS in the cloud, they are often geared towards particular domains such as bioinformatics [4, 5], astronomy [6], or can run workflows of particular kinds such as parameter sweep workflows [7] or QoS-annotated workflows [8]. Besides, many systems provide limited support for resource provisioning either by depending on a third party software to choose and provision virtual resources [18, 20] or user to do the provisioning manually [6, 18]. Finally, such systems are often configured to work with specific cloud [19] or simulated environments [7, 21].

Such ad-hoc solutions do not address the challenges identified in Section II. There is a pressing need for a generic, implementation- and platform-independent architectural solution that would address the cloud-related challenges for building cloud-enabled SWFMSs.

To address this need, we propose a generic, technology-independent architecture of cloud-enabled SWFMS including its subsystems and their interactions. We also present our DATAVIEW system which delivers a specific implementation of our architecture.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we first identified four key challenges of running big data workflows in the cloud. Second, we proposed a generic implementation-independent system architecture that provides guidance for different implementations of big data-oriented SWFMSs in the cloud and addresses most of the challenges discussed. Third, we implemented the DATAVIEW system that delivers a specific implementation of our architecture and ran a big data workflow in Amazon EC2, FutureGrid Eucalyptus, and FutureGrid Openstack clouds to validate that our system architecture is platform-independent. In the future, we plan to explore workflow scheduling techniques in the cloud that take advantage of workflow profiling and runtime behavior analytics module. We will also create more large-scale big data workflows from the automotive domain as well as workflows from bioinformatics and physics.

VII. REFERENCES