BIG DATA WORKFLOWS: A REFERENCE ARCHITECTURE AND THE DATAVIEW SYSTEM

Andrey Kashlev, Shiyong Lu, Aravind Mohan
Eastern Michigan University, Wayne State University, Allegheny College
andrey.kashlev@emich.edu, shiyong@wayne.edu, amohan@allegheny.edu

Abstract

The big data era is here, a natural result of the digital revolution of the last few decades. The emergence of big data in virtually all areas of life raises a fundamental question - how can we turn large volumes of bits and bytes into insights and possibly values? The answer to this question is often hindered by three big data challenges: volume, velocity, and variety. While scientific workflows have been used extensively in structuring complex scientific data analysis processes, they fall short in meeting the three big data challenges on the one hand, and in leveraging the dynamic resource provisioning capability of cloud computing on the other hand. To address such limitations, we propose and develop the concept of big data workflow as the next generation of data-centric workflow technologies. In this paper we: 1) identify the key challenges for running big data workflows in the cloud; 2) propose a reference architecture for big data workflow management systems (BDWFMSs) that addresses these challenges, 3) develop DATAVIEW, a big data workflow management system, to validate our proposed reference architecture, 4) design and run two big data workflows in the automotive and astronomy domains to showcase applications of our DATAVIEW system.

Keywords: scientific workflow, big data, cloud.

1. INTRODUCTION

The big data era is here, a natural result of the digital revolution of the last few decades. Data are being generated by a myriad of devices and events, from credit card transactions and ad clicks, to fitness wristbands and connected vehicles. This data deluge raises a fundamental question - how can we turn large volumes of bits and bytes into insights, decisions, and possibly values? The answer to this question is often hindered by three big data challenges: volume, velocity, and variety.

Consider the driver behavior analysis problem, in which we need to determine a driver's insurance premium based on their last three year's driving history[1]. Such an analysis involves large volume of data (over 75Gb per driver per year for OpenXC data Kashlev & Lu (2014a) or 750Mb/sec for a self-driving car), and to be more accurate, needs to be performed in combination with other data, such as data about the environment in which the vehicle is operating (weather, traffic, hazardous situations, etc.), and the driver's past claims and accident reports. The analysis is complex: one needs to extract all relevant features of the driving behavior from the raw data, perform deep analysis of these features in the context of other data sources to determine the risk of the driver, and based on the risk and the price model of the insurance company, suggest a quote for a given driver.

This kind of data analyses are often performed using scientific workflows, which are widely recognized to be an important paradigm in the services computing field Zhang (2011); Tsalgatidou et al. (2006) as they allow data scientists to compose various heterogeneous services into data analysis pipelines. While scientific workflows have been used extensively in structuring complex scientific data analysis processes, they fall short in meeting the three big data challenges on the one hand, and in leveraging the dynamic resource provisioning capability of cloud computing on the other hand. As scientists need to process data of high volume, velocity, and variety, it is imperative to enable scientific workflows to use distributed computing and storage resources available in the cloud in order to run so called big data workflows Kashlev & Lu (2014b); Hoffa et al. (2008). A big data workflow is the computerized modeling and

[1]For example, State Farm uses telematics to monitor a driving behavior by scoring the driver on various parameters, such as acceleration, braking and cornering.

automation of a process consisting of a set of computational tasks with data interdependencies to process and analyze data of ever increasing in scale, complexity, and rate of acquisition.

Unlike scientific workflows run in traditional on-premise environments such as stand-alone workstations or grids, big data 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" big data workflow management systems (BDWFMSs).

Although several traditional scientific workflow management systems (SWFMSs) have been developed to use cloud resources, they are geared towards either a specific domain such as bioinformatics Abouelhoda et al. (2012); Emeakaroha et al. (2013) or astronomy Vöckler et al. (2011), or a particular type of workflows such as workflows with parameter sweep and data fragmentation parallelism de Oliveira et al. (2010), or workflows with time and cost QoS constraints for each task Wu et al. (2013). 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. In addition, these solutions do not consider the problem of moving big data products. Thus, a 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 BDWFMS called DATAVIEW that delivers a specific implementation of our architecture and present two case studies from the automotive and astronomy domains to validate our solution.

We have tested our DATAVIEW implementation of the proposed architecture in three different cloud environments - Amazon EC2, FutureSystems Eucalyptus and FutureSystems OpenStack IaaS clouds[2]. We ran a workflow analyzing 3 Gb of vehicle data in OpenXC format OpenXC (2016). 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 Arbitron (2009); Kashlev & Lu (2014a).

This paper extends our earlier work Kashlev & Lu (2014b) with the following additional contributions:

1. We have identified five additional (sub-)challenges for running big data workflows in the cloud.
2. We have introduced two new modules in the architecture of the *Cloud Resource Manager* subsystem that enable intelligent and lightweight dependency management using docker containers.
3. We have added the Elastic Resource Management module in our *Workflow Engine* subsystem that dynamically adjusts the amount of virtual resources during workflow execution on as-needed basis, which improves resource utilization.
4. We have conducted another case study in which we ran a montage big data workflow from the astronomy domain in parallel across 20 virtual machines in the Amazon EC2 to further validate our architecture.

2. 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 Tsalgatidou et al. (2006); Zhao et al. (2011). 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 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

[2]FutureSystems portal was previously called FutureGrid

system in the cloud, according to the "all-in-the-cloud" approach Zhao et al. (2011). The system is deployed on a virtual machine in the cloud (master node) and is accessed remotely through a Web-based GUI interface. BDWFMS schedules workflows to run on multiple virtual machines (slave nodes) such that different parts of workflows run on different nodes to enable parallel execution. To start executing a workflow, BDWFMS provisions an appropriate amount of virtual machines that it will use to run the workflow. At the end of workflow execution, the slave nodes are terminated. We now describe major cloud-related challenges and their impact on scientific workflow management in the cloud.

2.1 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.

2.1.1 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 *eucarc* and *novarc*), and loading client software (e.g., *euca2ools* and *novaclient*), as in the case of both Eucalyptus and Openstack clouds FutureSystems (2017). 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.

2.1.2 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 EC2 (2017) provides Java API tools to manage its cloud resources programmatically, OpenStack OpenStack (2017) provides RESTful interface as well as python and command line implementations of OpenStack Nova API.

2.1.3 Creating machine images

The process of bundling, uploading, and registering images also varies across 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.

2.1.4 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 fifty seven on-demand instance types, while Openstack by default offers six instance types (called flavors). The types of instances the user had employed in the original cloud may not be supported by the target cloud. 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.

2.2 Resource Selection Challenge

Deciding on and provisioning appropriate amount of resources for a given workflow is a challenging task. Domain scientist needs to perform this task not only initially, upon creating the workflow, but also when rerunning the workflow with a different set of input files and/or input parameters.

2.2.1 Initial resource selection

Running a workflow in the cloud requires user (i.e. a domain scientist) to make a choice of the number and types of virtual machines to execute the workflow. Given a particular configuration (e.g., four *m1.xlarge*, seven

m3.xlarge, 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.

The blue boxes represent computational components (i.e. tasks), while the yellow boxes denote data products, which in this case are files. If the user chooses to run this workflow in Amazon EC2 cloud using three virtual servers, there are 57^3 = 185,193 possible choices for instance types for the three servers, since EC2 offers 57 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).

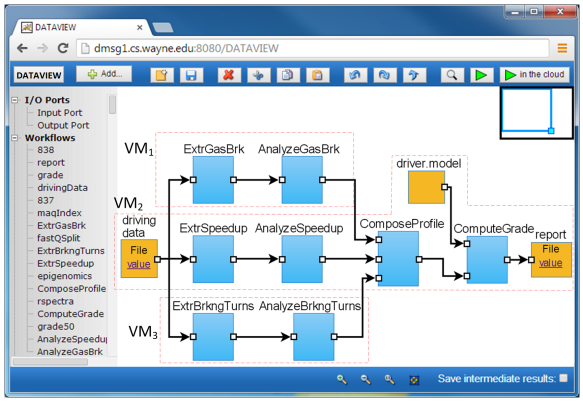


Figure 1: *Big data workflow analyzing automotive data.*

2.2.2 Resource selection when re-running the workflow

After a successful workflow execution, scientist may often need to re-run the workflow with a different set of input data products, such as files, and/or a different set of input parameters, as is the case in parameter sweep workflows. To re-run the workflow it may often be necessary to use more resources, e.g., if input files are larger, or if shorter makespan is desired. Determining what kind of new resources must be provisioned to achieve a given performance objective is a complicated task, e.g., how many new VMs to create and what type each VM should be to decrease the makespan by 20%. For example, if a workflow in Fig. 1 has been executed using three *m1.xlarge* virtual machines, adding the fourth VM of type *m1.xlarge* will clearly not improve workflow performance, since one of the four VMs will remain idle throughout the entire workflow execution.

2.3 Resource Utilization Challenge

Consider the Montage workflow from astronomy domain shown in Fig. 2. The workflow consists of multiple components (shown in blue), that analyze data to produce a mosaic of a set of sky images. Many data-intensive tasks in the workflow, including *mProjectPP*, *mDiffFit*, *mFitExec* and *mBgExec* are executed in parallel, in different virtual machines. For example, five instances of *mDiffFit* process different data products independently by being executed on five different VMs. This allows to reduce the workflow execution time, often referred to as *makespan*. Note, that the degree of parallelism of the workflow varies at different stages of workflow execution. It starts with four parallel branches (*mProjectPP*), then increases to five (*mDiffFit*), before decreasing to four *mBgExec*, and finally to one (*mAdd* and *mJPEG*). Fully taking advantage of this parallelism requires using five VMs for executing the workflow. However, only four out of five VMs will be used while running *mProjectPP* and *mBgExec* tasks. Moreover, only one out of five VMs will do useful work when executing *mAdd* and *mJPEG* components. Needless to say, that the user continues to pay for all five VMs, including those that are idle. Thus, leveraging workflow parallelism by executing independent branches in separate virtual machines has a side effect of poor resource utilization. Due to the fact that provisioning virtual machine takes time (often 30s and sometimes more), it is difficult to quickly add VMs on as-needed basis without introducing a delay in the workflow execution. We define VM utilization U_{VM} for a given period of time t as follows

$$U_{VM} = \frac{\sum_{i=1}^{n} A_i}{\sum_{i=1}^{n} A_i + \sum_{i=1}^{n} I_i} \quad (1)$$

where A_i denotes the total duration, in seconds, that a virtual machine VM_i was *active*, i.e. was performing computations, and I_i refers to the total time when VM_i was idle. When all the provisioned VMs are performing computations for the entire duration of t, $U_{VM} = 1$.

Besides VMs, it is sometimes difficult to track storage volumes that are no longer needed, which leads to needless expenses. For example, an intermediate data product can be saved in a large file and placed on an EBS storage volume. After such file is consumed by the downstream

component, neither the file nor the EBS volume are ever accessed again during workflow execution. Paying for such storage volumes to keep intermediate results for the entire duration of the workflow leads to unnecessary expenses. For example, in our montage workflow, the output file produced by the *mBgExec* task may be saved on an EBS volume attached to the VM where *mBgExec* executes. Once the file is sent to the VM where *mAdd* component runs, the EBS volume and its contents are no longer needed and hence can be deleted to save the cost.

2.3.1 Resource reusability

Reusing spare resources for running workflows is an important aspect of cloud-based workflow management. Spare virtual resources may appear after or during workflow run(s). When a workflow execution completes, the virtual machines used for running this workflow become idle and can be reused or terminated. Besides, spare resources may appear even before workflow finishes executing. For example, consider a workflow in Fig. 1, scheduled to run on three virtual machines, VM_1, VM_2, and VM_3. Upon completion of three parallel branches, the output files produced by *AnalyzeGasBrk* and *AnalyzeBrkngTurns* components are sent to VM_2, leaving VM_1 and VM_3 idle for the rest of the workflow execution. Thus, VM_1 and VM_3 can now be terminated or reused for running other workflows.

Reusing such VMs for running new workflows may 1) save time, as there will be no need to wait while the new VMs are being provisioned, 2) save cost, in case if VMs have been prepaid (e.g., in AWS VMs are paid for by hour without prorating the cost if terminated earlier). However, reusing such virtual resources is complicated for the following reasons.

1. It is challenging to configure existing VMs to satisfy all the dependencies of the new workflow, e.g., required libraries, software packages, environment variables, etc. For example, if a scientist wants to reuse existing VMs to run the astronomy workflow shown in Fig. 2, one must install montage software on these VMs, to be able to run *mProjectPP*, *mDiffFit*, *mAdd*, and other image processing components specific to astronomy domain.

2. It is often hard to reuse a set of existing virtual machines while ensuring the desired workflow performance, especially if some of these VMs are located in different regions (geographic locations), which can introduce latency due to limited network bandwidth. Sometimes, terminating some of the existing VMs and provisioning new VMs in the same region will help faster execute the workflow due to a superior network performance. It is a challenge to accurately determine which VMs should be terminated/replaced and which VMs can be readily reused for running a new workflow.

2.4 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.

2.4.1 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 BDWFMS 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 âĂŞ 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 BDWFMS and used for composing new workflows. While on a single machine addition of a new component is only performed once, for a BDWFMS 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.

2.4.2 Cataloging virtual resources

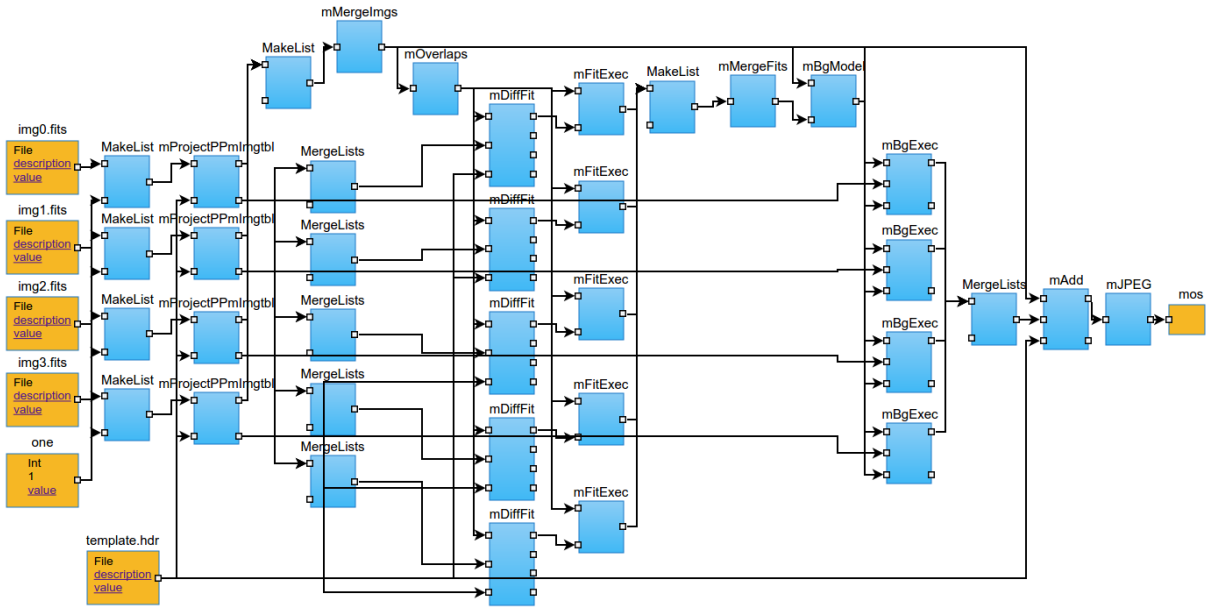


Figure 2: *Montage workflow for creating a mosaic of sky images.*

Running workflow 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 VMs 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.)

2.4.3 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 VMs 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 VMs suitable to execute workflow components is a challenge, given unique dependencies of each component.

2.5 Distributed Computing Challenge

The fact that the workflow execution is performed

in a distributed manner complicates big data workflow management in several ways.

2.5.1 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.

2.5.2 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 a 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.

2.5.3 Workflow debugging

In the event if a workflow execution fails, the need arises to backtrack the execution path to determine the cause of a failure, with the goal of re-running the workflow. The fact that workflow components execute in different virtual machines and send their data products across network makes debugging complicated. For example, it is common for a workflow execution to fail when one of the processes attempts to save file that does not fit on the disk. Diagnosing such failures is challenging as the error messages are often hard to find.

2.5.4 Fault tolerance

Enabling automated fault-tolerance capabilities, such as smart re-runs, is challenging for two reasons:

1. Given the distributed nature of cloud-based workflow execution, often across dozens and even hundreds of virtual machines, it is difficult to capture which parts of workflow have successfully finished.
2. Re-running a failed workflow will lead to an error again, unless appropriate changes are made to address the original cause of the workflow failure. For example, a new storage volume must be created and attached to a virtual machine, if a workflow failed due to a lack of storage space in this VM. Determining what changes must be made to ensure successful workflow re-run and performing such changes in an automated manner represents a great challenge.

2.5.5 Provenance collection

Since different components generally execute inside different virtual machines, collecting and storing the data derivation history of the entire workflow, while providing query and browsing interfaces, is a challenge.

3. A System Architecture for BDWFMS in the Cloud

We now present our proposed BDWFMS architecture, implemented in the DATAVIEW system, shown in Fig. 3. 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 Zhao et al. (2011), DATAVIEW system runs in the master node (see Fig. 3d). 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

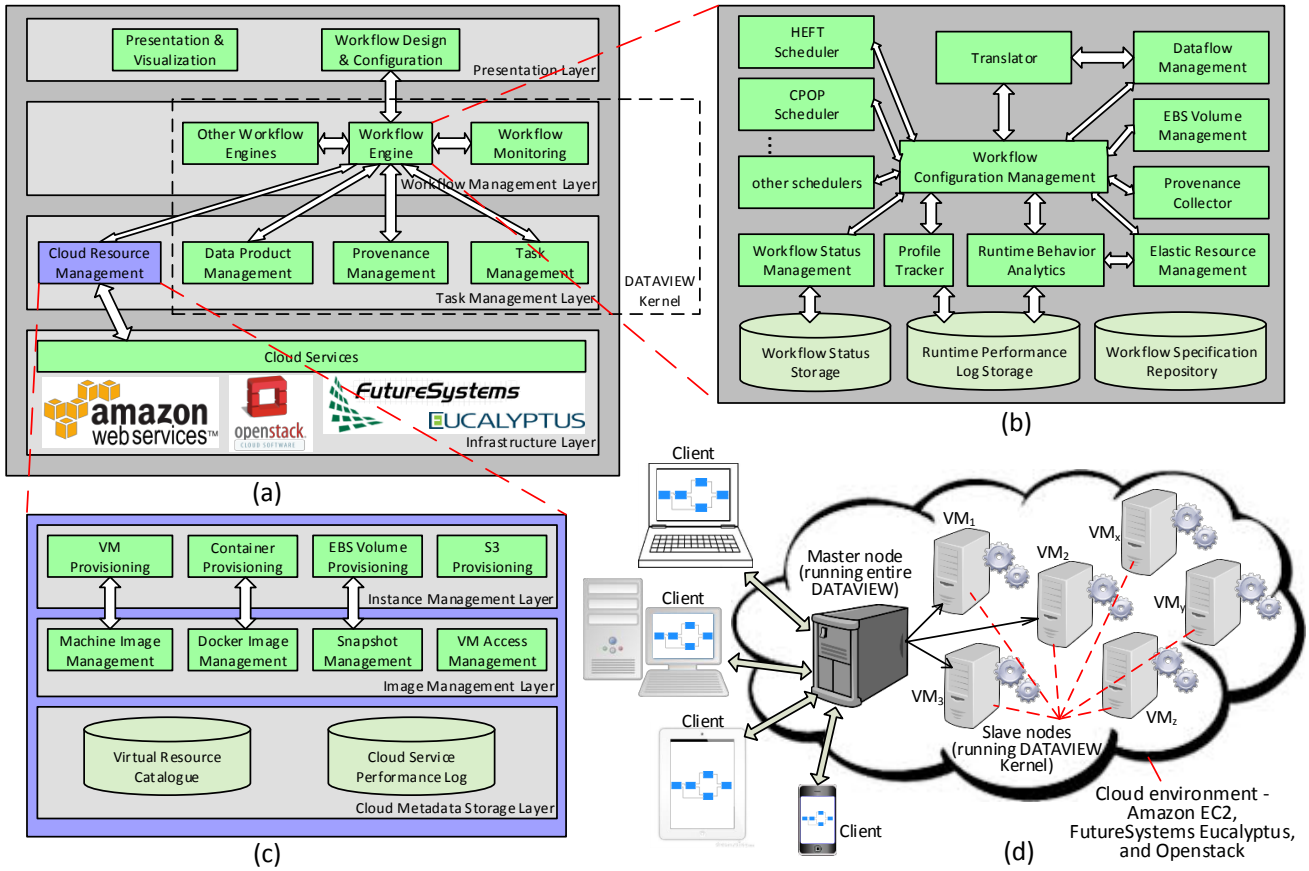


Figure 3: *(a) System architecture for big data-oriented BDWFMS in the cloud and zoom-in views of its two subsystems: (b) Workflow Engine, and (c) Cloud Resource Manager. (d) All-in-the-cloud deployment architecture of DATAVIEW.*

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, namely platforms heterogeneity challenge (connecting to the cloud and resource provisioning), as well as resource selection challenge. 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. 3b. 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. At runtime, the Workflow Configuration Management module stores the schedule. For example, the schedule for the workflow shown in Fig. 1 is as follows:

```
{
    "Component2VMmap":{
        "ExtrGasBrk":"VM1",
        "AnalyzeGasBrk":"VM1",
        "ExtrSpeedup":"VM2",
        "AnalyzeSpeedup":"VM2",
        "ExtrBrkngTurns":"VM3",
        "AnalyzeBrkngTurns":"VM3",
        "ComposeProfile":"VM1",
        "ComputeGrade":"VM1"
    },
    "dependencies":[
        "ComputeGrade":"Apache Mahout 0.9"
    ]
}
```

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 are 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 EC2 (2017) 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. 3d VM_1, VM_2, and VM_3, as shown in Fig. 1. Suppose the *AnalyzeGasBrk* component produced a large output file on VM_1 that needs to be moved to the VM_2 where *ComposeProfile* is scheduled to execute. Instead of sending a large file over the network, the system attaches an EBS volume to VM_1, stores output of *AnalyzeGasBrk* on that volume, detaches the volume from VM_1, and attaches the volume to VM_2, avoiding copying the file over the network altogether. Thus, the EBS Volume Management addresses the distributed computing 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 also addresses the distributed computing 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.

The Provenance Collector captures data derivation history in appropriate format such as OPMO OPM (2016) and sends it to the Provenance Manager to be stored. This addresses the provenance collection aspect of the distributed computing challenge.

The Elastic Resource Management (ERM) module intelligently requests to provision and terminate virtual resources (such as VMs and storage volumes) before, during, and after workflow execution, based on the workflow schedule, i.e. based on the current needs of the workflow. As the need to provision additional resources or terminate existing idle resources may arise during workflow execution, the ERM module consults with Runtime Behavior Analytics to determine the optimal time to send the provisioning/termination request. For example, consider the montage workflow shown in Fig. 2. As the workflow execution proceeds, the number of parallel branches in the workflow changes from four (initially), to five, to four, and to one. The user chooses to run the workflow with five virtual machines. The ERM module initially provisions four VMs (VM_1, VM_2, VM_3, and VM_4), before adding a fifth VM (VM_5). During the workflow execution, ERM requests provisioning of the fifth VM (VM_5), before *mDiffFit* is ready to execute, to account for the time it takes to provision VM_5. ERM module relies on the information provided by Runtime Behavior Analytics to determine at what point in time to send the provisioning request for VM_5. This is done to avoid a pause in workflow execution. Once the *mFitExec* task completes, Runtime Behavior Analytics determines whether it is beneficial to terminate only VM_5 and keep VM_1, VM_2, VM_3, and VM_4 for the sake of executing four instances of *mBgExec* in parallel, or to terminate all VMs except VM_1 and provision three new VMs once the execution reaches to *mBgExec*. Once *mBgExec* completes, all VMs except VM_1 are terminated. Dynamically increasing and decreasing the amount of virtual resources in this way allows to save cost during a workflow execution.

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 logging/monitoring aspect of the distributed computing challenge. The workflow monitoring information captured from executing a sample workflow, with ID = w89105, through the smart re-run option in the DATAVIEW system looks as follows:


```
{
   "workflowID-w89105":{
      "status":"executing",
      "cloud-resources":"192.10.5.24,
      192.10.5.101, 192.12.6.41,
      192.12.6.43, 192.12.6.55",
      "tasks-finished":"T1,T2,T3,T10,T25",
      "tasks-failed":"T21, T33, T59, T62",
      "tasks-inprogress":"T50,T52,T53",
      "data-produced":"T1.o1, T2.o1,
      T2.o2, T3.o1, T3.o2, T3.03, T10.o1,
```

```
      T25.o1, T50.o1, T52.o1, T53.o1"
   }
}
```

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 distributed computing challenge (passing data products to consumer components).

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. 3c.

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 resource volatility 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., and looks as follows:

```
{
   "ami-f1536798":{
      "os":"Ubuntu server x64 12.04",
      "provider":"aws",
      "platform":"ec2",
      "dependencies":[
         "python 3.3.3",
         "Apache Mahout 0.9",
        ...
      ]
   },

   "emi-1C8C3ADF":{
      "os":"Red Hat Linux",
      "provider":"futuregrid",
      "platform":"eucalyptus",
      "dependencies":[
         "perl 5.18.2",
         "R 3.0.2",
         ...
      ]
   },
   ...
}
```

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 resource volatility challenge (environment setup).

While VM images provide a reliable solution for managing dependencies, in many cases, it is possible to package software components in lightweight containers, managed by the docker platform Docker (2017). Docker containers package a piece of software, including code, runtime, system tools and libraries, i.e. everything that is needed for successful execution. Thus, docker containers guarantee that the software will run in any environment that has the docker platform installed. To leverage the docker platform for dependency management, we propose two modules - Docker Image Management and Container Provisioning. The Docker Image Management module allows to create lightweight docker images, capturing all the dependencies and libraries that a workflow component relies on. The docker images are often orders of magnitude smaller in size than the equivalent VM images. For example, the base Ubuntu image available on the Docker Hub, a registry for docker images, is only 188 Mb in size. In contrast, the size of an Ubuntu image available in Amazon EC2 cloud is 8Gb.

Consider a workflow in Fig. 2 that creates a mosaic of sky images. Each component of this workflow relies on montage software for performing its computations. Whenever a component is scheduled to run on a VM, the Container Provisioning module will create a docker container inside this VM, using appropriate docker image, i.e. one that contains montage software. Thus, the workflow component can execute successfully. This eliminates the need to create a large VM image. In the

context of running big data workflows in the cloud, using docker images for managing dependencies provides three important advantages:

1. Lightweight support of a large number of diverse components that require a broad range of libraries, software packages, and environment variables.

2. Better reuse of idle virtual machines. If a spare VM is available that lacks certain dependencies to run a workflow component, container(s) featuring the required packages can be deployed in this VM to enable its reuse.

3. Isolation of each component's dependencies. Multiple docker containers can be deployed inside a VM, with each container's dependencies being fully isolated from other containers. This is crucial when two or more components with conflicting sets of dependencies are scheduled to run on the same VM. Some components, for example, may require different versions of python. Therefore, using docker images also improves resource utilization.

In situations when docker containers cannot be employed (e.g., for Windows-based workflow components), the traditional VM images must be used.

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 distributed computing challenge (passing data products to consumers). 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 are still available. This addresses the resource volatility challenge (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.

4. 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 deployed DATAVIEW in Amazon EC2 EC2 (2017) as well as the Futuregrid's Eucalyptus and Openstack FutureSystems (2017). We ran several workflows in these cloud environments. As the results were similar in different cloud environments, due to space limit here we report the results obtained in the Amazon EC2.

We ran a big data workflow from the automotive domain in the Amazon EC2 cloud. The implementation and case study show how our architecture addresses the platform heterogeneity, resource volatility, and distributed computing challenges. We are extending our system functionality to address the resource volatility challenge.

4.1 DATAVIEW Implementation in the Cloud

DATAVIEW is based on our earlier general purpose SWFMS called VIEW. It extends VIEW with additional functionality that enables workflow execution in the cloud environment, including the new *Workflow Engine* and *Cloud Resource Manager* subsystems.

Fig. 1 shows the Web-based GUI on DATAVIEW. The sidebar on the left-hand side lists the executable workflows, as well as reusable workflows (i.e. tasks, or building blocks). The workflow itself is built by dragging-and-dropping reusable components onto the DATAVIEW workspace.

Upon completing the workflow design via DATAVIEW's intuitive drag-and-drop interface, the user presses the "Provision VMs" button on the DATAVIEW toolbar to allocate virtual machines for workflow execution. The user specifies the number of VMs, and the type of each machine using a dialog shown in Fig. 4.

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.

4.2 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 OpenXC (2016).

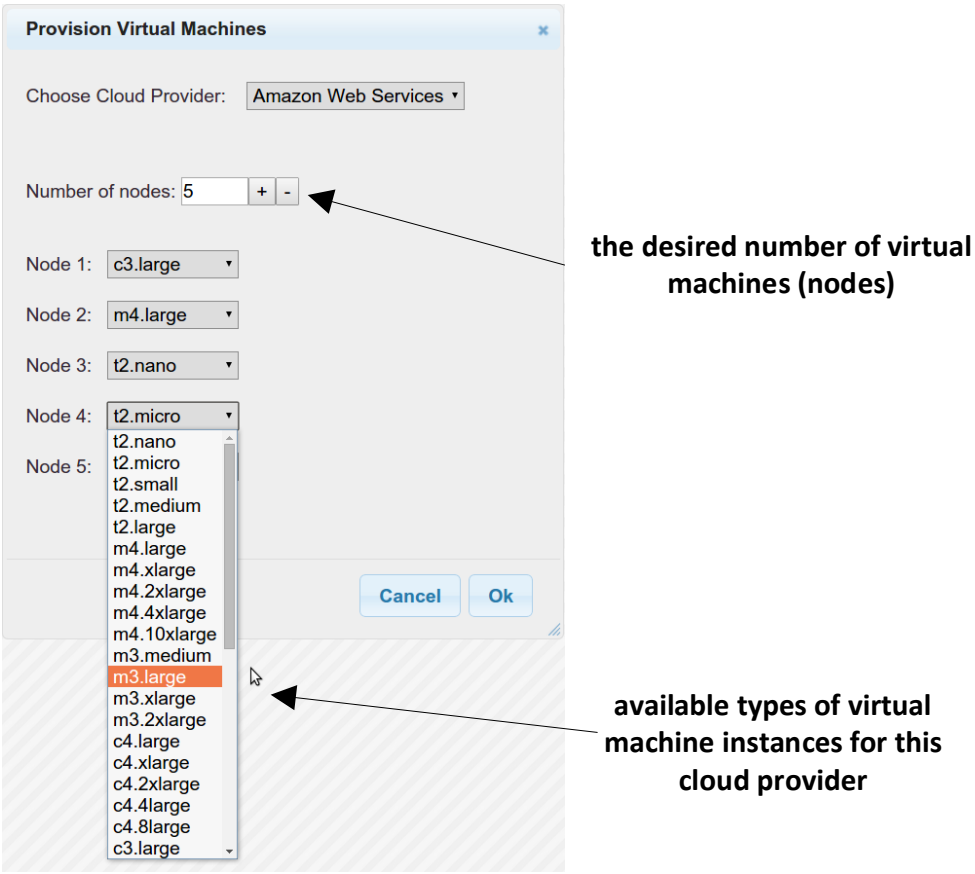
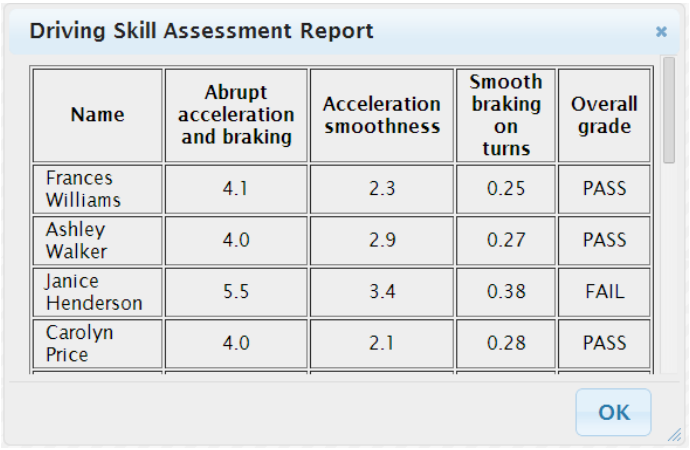


Figure 4: *Provisioning virtual machines in DATAVIEW.*

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 OpenXC (2016).

Our dataset is equivalent to 1 hour worth of data, collected from 50 drivers making the size of the input file 3Gb Kashlev & Lu (2014a). 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. It then 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 competenency test and produces a final result of the workflow – driving skill assessment report, which is displayed in a pop-up window by DATAVIEW (Fig. 5). 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



Name	Abrupt acceleration and braking	Acceleration smoothness	Smooth braking on turns	Overall grade
Frances Williams	4.1	2.3	0.25	PASS
Ashley Walker	4.0	2.9	0.27	PASS
Janice Henderson	5.5	3.4	0.38	FAIL
Carolyn Price	4.0	2.1	0.28	PASS

Figure 5: *Screenshot of the driving skill assessment report produced by our big data workflow in the DATAVIEW system.*

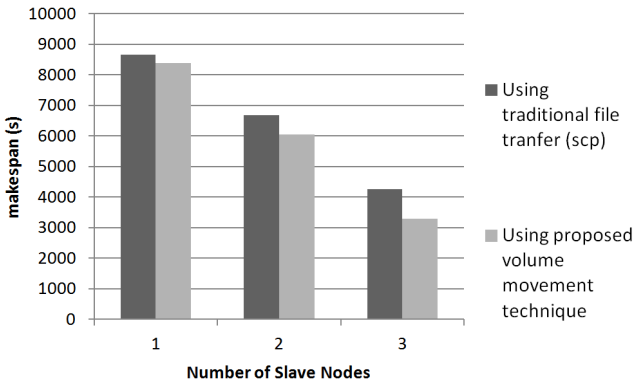


Figure 6: *Running big data workflow from the automotive domain in Amazon EC2 with different number of nodes.*

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. 6 we report the performance study results from running our scientific workflow in the Amazon EC2. Our system used the HEFT algorithm Topcuoglu et al. (2002) to schedule the workflow onto the VMs. As shown in Fig. 6, 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.

4.3 Case Study: Building Sky Image Mosaic

We designed and ran montage workflow from the astronomy domain[3], shown in Fig. 2. We ran the workflow

[3]This research made use of Montage. It is funded by the National Science Foundation under Grant Number ACI-1440620, and was previously funded by the National Aeronautics and Space Administration's Earth Science Technology Office, Computation Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA

in the Amazon EC2 cloud using our DATAVIEW system. We successfully executed the montage workflow in a distributed fashion across 20 virtual machines. While executing in parallel across these 20 virtual machines, our workflow has processed and reprojected 4.2 Gb of astronomical image data. The workflow execution time was 67s. The workflow creates a mosaic of astronomical images in the popular *.fits* format. The *MakeList* component helps wrap a set of images in a list structure. First, the *mProjectPP* component reprojects each image to the scale defined in the FITS header template (*template.hdr* file). Next, the *mImgTbl* component extracts the FITS header geometry information from a set of files and creates an ASCII image metadata table which is used by several of the other programs. The *mProjectPPmImgtbl* component, which consists of *mProjectPP* and *mImgtbl*, produces a pair of images: the reprojected image and an "area" image. The area image goes through all the subsequent processing that the reprojected image does, allowing it to be properly coadded at the end. Once the *images.tbl* file has been produced by *mMergeImgs*, the *mOverlaps* component analyzes the image metadata table to determine a list of overlapping images. Each image is compared with every other image to determine all overlapping image pairs. A pair of images are deemed to overlap if any pixel around the perimeter of one image falls within the boundary of the other image. The result is the *diffs.tbl* file. Next, the *mDiffFit* function is called to calculate the difference between a single pair of overlapping images, and to fit a plane to an image using least squares. After this, *mDifExec* creates a table of image-to-image difference parameters, stored in the *fits.tbl* file. The *mBgModel* function uses the image-to-image difference parameter table created by *mDifExec* to interactively determine a set of corrections to apply to each image in order to achieve "best" global fit. *mAdd* coadds the reprojected images in an input list to form an output mosaic with FITS header keywords specified in a header file. It creates two output files, one containing the coadded pixel values, and the other containing coadded pixel area values. Finally, the *mJPEG* function generates a JPEG image file from the *.fits* file produced by *mAdd*.

4.4 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. 7.

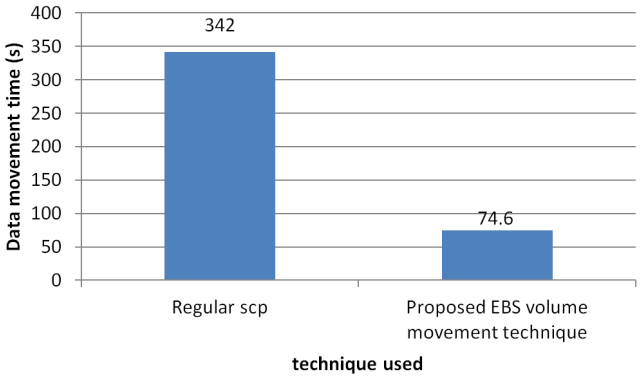


Figure 7: *Moving 3Gb dataset to the target VM.*

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. 6), since more nodes require more data movement. For more data-intensive workflows such performance gain is even larger.

5. Related Work

In the age of big data the unprecedented access to on-demand computing resources makes cloud an essential paradigm, as it enables everyone, even small research teams, to dynamically build virtualized cyberinfrastructures for running their large-scale scientific workflows in order to extract knowledge and value from big data. The need to utilize cloud computing to run scientific workflows has been widely recognized by the scientific community Zhao et al. (2011); Deelman (2015); Zhao et al. (2015); de Oliveira et al. (2012). Many researchers studied and confirmed the feasibility of using cloud computing for e-science from both cost Deelman et al. (2008) and performance perspectives Iosup et al. (2011); de Oliveira et al. (2013).

and the California Institute of Technology.

In Deelman (2010), 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 Zhao et al. (2011). Ostermann and Prodan Ostermann & Prodan (2012) analyze the problem of provisioning Cloud instances to large scientific workflows, and show how Spot instances can be used for scientific workflow execution. Juve et al. examine the performance and cost of clouds when executing scientific workflow applications, and consider three different workflows of different I/O, Memory, and CPU intensity.

A number of efforts were made towards building systems for running scientific workflows in the cloud. In Vöckler et al. (2011), 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 Wu et al. (2013), 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. de Oliveira et al. (2010) present SciCumulus, a cloud middleware that explores parameter sweep and data fragmentation parallelism in scientific workflow activities. The authors present a conceptual architecture geared towards parameter sweep and data fragmentation and run their system in the simulated cloud. In Abouelhoda et al. (2012) Abouelhoda et al. propose a system called Tavaxy that allows seamless integration of the Taverna system with Galaxy workflows based on hierarchical workflows and workflow patterns. Tavaxy has an interface to set up a cluster in AWS cloud and use it to run workflows. Wang et al. Wang & Altintas (2012) report preliminary work and experiences of enabling the interaction between Kepler SWFMS and the EC2 cloud. In Vahi et al. (2013), Vahi et al. discuss usage of object stores and shared file systems for managing data products in big data scientific workflows.

While these solutions provide some insights into development of SWFMSs in the cloud, they are often geared towards particular domains such as bioinformatics Abouelhoda et al. (2012); Emeakaroha et al. (2013), astronomy Vöckler et al. (2011), or can run workflows of particular kinds such as parameter sweep workflows de Oliveira et al. (2010) or QoS-annotated workflows Wu et al. (2013). Besides, many systems provide limited support for resource provisioning either by depending on a third party software to choose and provision virtual resources Deelman (2010); Pandey et al. (2011) or user to do the provisioning manually Vöckler et al. (2011); Deelman (2010). Finally, such systems are often configured to work with specific cloud Wang & Altintas (2012) or simulated environments de Oliveira et al. (2010); Yuan et al. (2010).

These solutions, many of which are ad hoc in nature, do not address the breadth of challenges that we identify. 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 BDWFMSs.

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

6. Conclusions and Future Work

In this paper, we first identified five 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 BDWFMSs 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 two big data workflows in Amazon EC2 cloud to validate our system architecture. 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 Internet of Things (IoT) domains.

Acknowledgement

This work is supported by U.S. National Science Foundation under ACI-1443069 and is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

References

Abouelhoda, M., Issa, S., & Ghanem, M. (2012). Tavaxy: Integrating taverna and galaxy workflows with cloud computing support. *BMC Bioinformatics*, *13*, 77.

Arbitron (2009). *The Arbitron national in-car study*. Arbitron Inc.

de Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E. S., Dias, J., de A. R. Gonçalves, J. C., Baião, F. A., & Mattoso, M. (2013). Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows. *Future Generation Comp. Syst.*, *29*(7), 1816–1825.

de Oliveira, D., Ogasawara, E. S., Baião, F. A., & Mattoso, M. (2010). SciCumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *IEEE International Conference on Cloud Computing, CLOUD 2010, Miami, FL, USA, 5-10 July, 2010*, (pp. 378–385).

de Oliveira, D., Ogasawara, E. S., Ocaña, K. A. C. S., Baião, F. A., & Mattoso, M. (2012). An adaptive parallel execution strategy for cloud-based scientific workflows. *Concurrency and Computation: Practice and Experience*, *24*(13), 1531–1550.

Deelman, E. (2010). Grids and clouds: Making workflow applications work in heterogeneous distributed environments. *IJHPCA*, *24*(3), 284–298.

Deelman, E. (2015). Challenges of running scientific workflows in cloud environments. In *Proceedings of the 6th Workshop on Scientific Cloud Computing, ScienceCloud 2015, Portland, Oregon, USA, June 16, 2015*, (p. 1).

Deelman, E., Singh, G., Livny, M., Berriman, G. B., & Good, J. (2008). The cost of doing science on the cloud: the montage example. In *Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2008, November 15-21, 2008, Austin, Texas, USA*, (p. 50).

Docker (2017). *The Docker Virtualization Software*. `https://www.docker.com/`.

EC2 (2017). *Amazon EC2 Cloud*. `https://aws.amazon.com/ec2/`.

Emeakaroha, V. C., Maurer, M., Stern, P., Labaj, P. P., Brandic, I., & Kreil, D. P. (2013). Managing and optimizing bioinformatics workflows for data analysis in clouds. *J. Grid Comput.*, *11*(3), 407–428.

FutureSystems (2017). *FutureSystems portal*. `https://portal.futuresystems.org/`.

Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, G. B., & Good, J. (2008). On the use of cloud computing for scientific workflows. In *Fourth International Conference on e-Science, e-Science 2008, 7-12 December 2008, Indianapolis, IN, USA*, (pp. 640–645).

Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., & Epema, D. H. J. (2011). Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans. Parallel Distrib. Syst.*, *22*(6), 931–945.

Kashlev, A., & Lu, S. (2014a). A system architecture for running big data workflows in the cloud. Tech. Rep. TR-BIGDATA-10-2013-KLC, Wayne State University. Available from `http://www.cs.wayne.edu/andrey/papers/TR-BIGDATA-02-2014-KL.pdf`.

Kashlev, A., & Lu, S. (2014b). A system architecture for running big data workflows in the cloud. In *IEEE International Conference on Services Computing, SCC 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, (pp. 51–58).

OpenStack (2017). *OpenStack Cloud Software*. `https://www.openstack.org/`.

OpenXC (2016). *The OpenXC Platform*. `http://openxcplatform.com/`.

OPM (2016). *The Open Provenance Model*. `http://openprovenance.org`.

Ostermann, S., & Prodan, R. (2012). Impact of variable priced cloud resources on scientific workflow scheduling. In *Euro-Par 2012 Parallel Processing - 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings*, (pp. 350–362).

Pandey, S., Karunamoorthy, D., & Buyya, R. (2011). Workflow engine for clouds. *Cloud Computing: Principles and Paradigms*, (pp. 321–344).

Topcuoglu, H., Hariri, S., & Wu, M. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, *13*(3), 260–274.

Tsalgatidou, A., Athanasopoulos, G., Pantazoglou, M., Pautasso, C., Heinis, T., Grønmo, R., Hoff, H., Berre,

A., Glittum, M., & Topouzidou, S. (2006). Developing scientific workflows from heterogeneous services. *SIGMOD Record*, *35*(2), 22–28.

Vahi, K., Rynge, M., Juve, G., Mayani, R., & Deelman, E. (2013). Rethinking data management for big data scientific workflows. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, (pp. 27–35).

Vöckler, J.-S., Juve, G., Deelman, E., Rynge, M., & Berriman, B. (2011). Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, (pp. 15–24). ACM.

Wang, J., & Altintas, I. (2012). Early cloud experiences with the Kepler scientific workflow system. In *Proceedings of the International Conference on Computational Science, ICCS 2012, Omaha, Nebraska, USA, 4-6 June, 2012*, (pp. 1630–1634).

Wu, Z., Liu, X., Ni, Z., Yuan, D., & Yang, Y. (2013). A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, *63*(1), 256–293.

Yuan, D., Yang, Y., Liu, X., & Chen, J. (2010). A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Conference Proceedings*, (pp. 1–12).

Zhang, L. (2011). Editorial: Quality-driven service and workflow management. *IEEE T. Services Computing*, *4*(2), 84.

Zhao, Q., Xiong, C., Zhao, X., Yu, C., & Xiao, J. (2015). A data placement strategy for data-intensive scientific workflows in cloud. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, (pp. 928–934).

Zhao, Y., Fei, X., Raicu, I., & Lu, S. (2011). Opportunities and challenges in running scientific workflows on the cloud. In *2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2011, Beijing, China, October 10-12, 2011*, (pp. 455–462).

Authors:

Andrey Kashlev is an Assistant Professor in the Department of Computer Science at Eastern Michigan University. He is conducting research in big data management with the focus on big data modeling and large-scale scientific workflows. His broader interests include big data, databases, data analytics, data science, provenance, big data curation, Semantic Web, Internet of Things, and services computing. He has completed his Ph.D. in Computer Science at Wayne State University in 2016 in the Big Data Research Lab led by Dr. Shiyong Lu. He has published numerous research articles in peer-reviewed international journals and conferences, including IEEE Transactions on Services Computing (TSC), Data and Knowledge Engineering (DKE), International Journal of Computers and Their Applications (IJCA), IEEE International Congress BigData (IJBD), and IEEE International Conference on Services Computing (SCC), and others.

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. Dr. Lu received his Ph.D. in computer science from Stony Brook University in 2002. Dr. Lu's current research interests focus on big data workflows, big data modeling, and provenance management. Dr. Lu is an author of two books and over 120 articles published in various international journals and conferences, including IEEE Transactions on Services Computing (TSC), Data and Knowledge Engineering (DKE), IEEE Transactions on Knowledge and Data Engineering (TKDE). He is the founding chair of the IEEE International Workshop on Scientific Workflows (SWF) and a Co-Editor-in-Chief of the International Journal of Cloud Computing and Services Science. He is a founding editorial board member of International Journal of Big Data and a senior member of the IEEE.

Aravind Mohan is an Assistant Professor in the Department of Computer Science at Allegheny College. He is conducting research in big data management with the focus on NoSQL databases, cloud computing, and data intensive workflows. His broader interests include big data, databases, data analytics, data science, provenance, information retrieval, online education platforms, Internet of Things, and services computing. He has completed his Ph.D. in Computer Engineering at Wayne State University in 2017 in the Big Data Research Lab led by Dr. Shiyong Lu. He has published several research articles in peer-reviewed international conferences and journals, including the IEEE international conference on services computing, big data congress, big data, big data computing services and applications, the ACM conference on SIGIR and ECIR, and the International Journal of Big Data. He is a member of IEEE and ACM.