# Automated Cluster Provisioning And Workflow Management for Parallel Scientific Applications in the Cloud


Brandon Posey
*School of Computing*
*Clemson University*
Clemson, USA
bposey@clemson.edu

Christopher Gropp
*School of Computing*
*Clemson University*
Clemson, USA
cgropp@clemson.edu

Alexander Herzog, Ph.D.
*School of Computing*
*Clemson University*
Clemson, USA
aherzog@clemson.edu

Amy Apon, Ph.D.
*School of Computing*
*Clemson University*
Clemson, USA
aapon@clemson.edu



***Abstract*—Many commercial cloud providers and tools are available that researchers could utilize to advance computational science research. However, adoption by the research community has been slow. In this paper we describe the automated Provisioning And Workflow (PAW) management tool for parallel scientific applications in the cloud. PAW is a comprehensive resource provisioning and workflow tool that automates the steps of dynamically provisioning a large scale cluster environment in the cloud, executing a set of jobs or a custom workflow and, after the jobs have completed, de-provisioning the cluster environment in a single operation. A key characteristic of PAW is that it separates the provisioning of cluster resources in the cloud from the management of scientific workflow on these resources, which enables fine-grained decisions about performance and cost tradeoffs in a commercial cloud environment. This paper describes our initial AWS implementation of PAW for executing a large parameter sweep workflow. We demonstrate this using an MPI-based topic modeling application. PAW provides a standardized, simplified, and pluggable interface that can easily be expanded to support a variety of underlying cloud or cluster hardware environments, user-facing scheduling systems, workflows, and scientific applications.**



***Index Terms*—HPC cluster, cloud computing, parallel scientific applications, dynamic resource provisioning, workflow**


## I. Introduction

The availability of computing, data, and analytics services through commercial cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) is growing rapidly. According to Gartner, infrastructure cloud services are projected to grow at a rate of more than 36% in 2017 [1]. However, the adoption of commercial clouds for parallel scientific computation has been slow for a number of reasons [2]. For example, many researchers do not have the time or access to the proper resources to learn how best to use commercial cloud resources for scientific applications. Managing funding and billing for cloud resources is challenging. Researchers may hesitate to use cloud resources because optimizing runtime configurations is complex with many resource options. Domain scientists seek a familiar interface and computing environment, with minimal setup and training, when considering use of the commercial cloud.

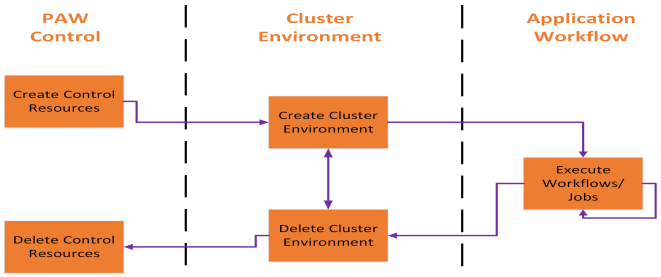


Fig. 1. Simplified View of PAW Operational Stages of Execution

The automated cluster Provisioning And Workflow (PAW) management tool described in this paper enables dynamic provisioning of a cluster environment in the cloud and supports management of complex scientific workflows in a familiar scientific environment. PAW allows researchers with minimal commercial cloud knowledge to submit and run custom workflows on the cloud with minimal modifications of existing job scripts/workflows, allowing researchers to focus on research rather than learning a new tool. PAW can build a familiar cluster resource for parallel applications and dynamically provision it using common, default settings for cloud resources. PAW can also provision resources for experienced users by allowing detailed specifications for computing and storage resources, depending on the scientific application and needs.

PAW begins to address billing challenges as described in [2]. While there is attraction to the cloud model of paying for resources only while they are being used, one of the risks in utilizing the commercial cloud is that resources left running longer than they are actually needed can increase costs unnecessarily. The automated de-provisioning of cluster environments provided by PAW helps to alleviate that risk.

PAW's key innovation is separating the provisioning of cluster resources in the cloud from the management of scientific workflow on these resources. Fig. 1 illustrates a simplified view of the PAW operational stages of Control, Environment, and Workflow. The separation of operational stages enables more fine-grained control over performance and cost tradeoffs in a commercial cloud environment than is otherwise possible.

PAW execution begins with the creation of *control resources* on the specified cloud platform (e.g., AWS), as shown on the left in Fig. 1. The role of the PAW control resources is to manage the automated provisioning of various *cluster environments*. More than one cluster environment can be provisioned, and these do not have to be the same as each other. PAW creates and provisions fully functional cluster environments using the specified cloud resources. Each configurable cluster environment includes typical resources such as login node, scheduler, parallel file system, and support for MPI applications. The cluster environment is accessible to the user via ssh for debugging, just as a campus cluster is accessible to a user via ssh to the login node. Upon the successful provisioning of the cluster environment, PAW can automatically submit the specified workflows and jobs to the provisioned environment. When the workflows and jobs complete, PAW automatically de-provisions and deletes the *cluster environment* and *control resources*. These resources can also optionally be left running for further application workflow submissions.

This paper describes our initial implementation of PAW using AWS and a case study of the use of PAW for an integrated large scale parameter sweep workflow. Our experiments demonstrate PAW on an MPI-based topic modeling application. Our results show how PAW provisions a large scale cluster environment on AWS, submits the application workflow and, upon completion of the workflow, de-provisions the cluster environment with no user interaction. We also describe how PAW can be applied to other commercial clouds and local on-premise clusters.

The remainder of this paper is organized as follows. Section II describes related work. Section III describes the supporting environment and tools utilized to build PAW. Section IV describes the three main architectural components of PAW: Initial Setup and Configuration, Workflow/Job Submission and Monitoring, and De-provisioning. In Section V we report PAW's scalability performance and our case study results. In Section VI we summarize our results and future work.

## II. Related Work

Tools that implement the full operational steps of PAW, including the dynamic provisioning of a cluster environment, submission and execution of workflows, and the deletion of the cluster environment, include AWS Batch, Galaxy CloudMan, elasticHPC, and HTCondor. In general, these tools provide resource provisioning along with workflow management for applications in a particular field of science and may require extra configuration or knowledge to use.

AWS Batch is an AWS managed service that enables users to run batch computing workloads in AWS [9]. A user packages the code for the jobs, specifies the dependencies, and submits the package to AWS Batch. AWS Batch automatically provisions compute resources and optimizes the workload distribution for the user. However, the user must set up an MPI programming environment and also the scheduler, if that is desired. To use AWS Batch a user has to rewrite the job scripts to run on AWS Batch. AWS Batch is based on Docker Containers and AWS's Elastic Container Service (ECS), which requires a pre-configured Docker image to utilize the service. Unlike PAW, AWS Batch does not allow login access to the instances for debugging or other troubleshooting, which can make finding and fixing bugs time consuming since the entire environment must be relaunched for every test run.

Galaxy's CloudMan tool provides some features similar to PAW but it is heavily customized for use by researchers in biology and genetics. CloudMan enables the provisioning of Galaxy, a web-based platform focusing on biomedical research, on a CloudMan cluster within an AWS, OpenStack, or OpenNebula cloud. The provisioned cluster can optionally be pre-configured with the Galaxy software suite. One version comes with the Slurm HPC scheduler, NFS storage, and interactive access [10]. However, in this mode, CloudMan does not manage submission of custom workflows. CloudMan works well for Galaxy workflows but is not a general-purpose workflow manager, so the audience is limited primarily to biology and genetic scientists.

Another similar tool is elasticHPC [11], which is designed for use in bioinformatics. It creates a cluster environment with an HPC scheduler, NFS shared storage, and pre-configured bioinformatics software. It has an interface to work with jobs but the process is not automated. The project does not seem to have been updated since 2012.

HTCondor is a software system that creates a high-throughput computing environment by utilizing the computing power of workstations that communicate over the network [12]. However, there are some limitations to the types of workflows that can be executed. HTCondor workflows are not able to do interactive input and output, the jobs must not be multi-process, and the jobs are not able to have interprocess communication. HTCondor provides an annex that allows it to interact with different cloud providers, however this is not packaged for the user and is meant for use by system administrators instead of end users.

There are also a variety of tools that have been developed for managing workflows that unlike PAW do not also manage or provision resources. Examples include Tigres [3], FireWorks [4], QDO [5], SWIFT [6], Pegasus [7], and DAGman [8]. Since these only provide part of the capability of PAW, we describe a couple of examples only. Tigres is a template library that allows researchers to quickly develop, test, and deploy analysis workflows through the use of templates [3]. Similar to other workflow management tools, Tigres does not provision any resources and assumes that all the resources required to run the workflow are already created. FireWorks is similar in that it does not provision resources but instead works like an HPC scheduler to manage different "FireTasks" that are assigned to different "FireWorkers" [4]. FireWorkers can be a local machine or resources managed by an existing HPC scheduler, but resources have to exist before FireWorks can be utilized.

PAW is not meant to replace workflow management tools, but to complement them and make them more usable in the commercial cloud. For example, a custom PAW workflow

could include the use of SWIFT to manage the computational workflow while PAW performs the resource management tasks. This is the power and flexibility behind the custom workflow interface within PAW. Existing tools can be integrated as custom workflows, which will allow PAW to become be used across a range of cloud workflows. Implementing and integrating custom PAW workflows using different standard workflow management tools is a topic of future work.

## III. Supporting Environment and Tools

PAW has been developed with as few external libraries and dependencies as possible to simplify the installation and configuration process. The initial implementation uses AWS services and CloudyCluster, as described in this section.

### A. Amazon Web Services (AWS)

AWS is the largest commercial cloud computing provider [13]. AWS provides access to certified solutions architects and cloud credits for research [14]. AWS services utilized by PAW include: EC2, DynamoDB, Simple Storage Service (S3), Elastic Filesystem (EFS), Identity and Access Management (IAM), Autoscaling, and CloudFormation. EC2 allows users to create a variety of virtual machines (VMs). DynamoDB is a NoSQL database. IAM is AWS's permission and security mechanism for managing the access that users have to services within an AWS account. The implementation uses Python 2.7 and two external libraries, Boto3 and Botocore, both of which are required for access to AWS Services. The only information required from the user for initial configuration is the user's AWS credentials for creating and deleting the required resources.

### B. CloudyCluster

A core component of PAW is CloudyCluster, which supports the dynamic provisioning and de-provisioning of cluster environments within commercial clouds [15], [16]. The provisioned cluster can include shared filesystems, NAT instance, compute nodes, parallel filesystem, login node, and schedulers.

PAW accesses most AWS services through CloudyCluster APIs, and utilizes these APIs to perform many administrative tasks. CloudyCluster provides several configuration options, such as the choice of scheduler and shared home directories. All environment customization within PAW is accomplished through templates. PAW comes pre-packaged with a CloudyCluster template generator and several standard CloudyCluster environment templates. The template generator allows users to create, share, and modify their own CloudyCluster environment templates.

After the user chooses a template, CloudyCluster executes the required AWS API calls to dynamically provision and configure the requested environment resources. CloudyCluster also pre-configures the selected scheduler (e.g., Torque or Slurm), creates users on the nodes, and mounts the requested filesystems on every node.

CloudyCluster provides a meta-scheduler called Cloudy Cluster Queue (CCQ) that provides job-driven autoscaling via either special directives within the job script or through parsing certain information from supported HPC scheduler directives [16]. The integration with CCQ and the job level autoscaling provided by CCQ are key features that make PAW different from the other utilities that provision cluster environments. Utilizing CCQ, PAW can submit a job and dynamically create the exact resources that the workflow and job require, and release the job to the HPC scheduler only when the resources have been successfully created. All of the job submission and communication that PAW performs with CCQ is done through the web based APIs.

CCQ enables cost management. After the workflow or job has completed and the resources are no longer performing useful work, CCQ analyzes how far into the current billing period the instance is. If the instance is close to the end of the billing period (e.g., one hour), the instance will be terminated since it is not being utilized. However, if the instance has not reached the end of its billing period, the instance will continue to run until it reaches the end of its billing period. This optimizes the amount of time that the instance is available for new workflows/jobs. However, if a new workflow/job is submitted and starts using an instance near the end of its billing period, the instance will not be killed and instead will continue to execute until the new workflow/job has been completed. This architecture helps to minimize the costs and the time required to start new workflows/jobs.

PAW is able to use AWS Spot pricing since both CloudyCluster and CCQ are able to utilize the AWS Spot market. The Spot market allows users to bid for unused capacity on AWS. Use of Spot can decrease costs, as the average spot prices for different instances types is up to 90% less than the on-demand price for the same instance type. PAW accesses the Spot market via a few simple directives that are added to existing job scripts. PAW also exposes this feature of CCQ and CloudyCluster through its configuration file, discussed later.

## IV. Technical Details of PAW

In this section we provide technical details about the architectural components and the operational stages of PAW.

### A. Architectural Components

The architecture of PAW is implemented in a way that allows for easy expansion and integration of components with other resources and services. A human readable *ini* formatted configuration file is central to the architecture of PAW. The *ini* format enables utilization of Python's built-in *ConfigParser* to read and process the configuration file. Other components of PAW's architecture are Resources, Environments and Environment Templates, Schedulers, and Workflow Templates. Each of these is described in this section.

*1) Configuration File:* PAW is driven by a *ini* style configuration file that defines the parameters required by PAW to execute. The configuration file has six different sections: `UserInfo`, `General`, `Cloud Type Settings`, `Environment Settings`, `Computation`, and `Environment Templates`. Of these six sections both

```
[UserInfo]
userName: bposey
password: Passw0rd2017

[General]
environmentName: topicModelingTest
cloudType: aws

[CloudyClusterAws]
keyName: bposey-key
instanceType: c3.8xlarge
networkCidr: 0.0.0.0/0
vpc: vpc-13a3f474
publicSubnet: subnet-12e8645b
capabilities: CAPABILITY_IAM
region: us-east-1

[CloudyClusterEnvironment]
templateName: noSharedFilesystem
keyName: myKeyFile
region: us-east-1
az: us-east-1a

[Computation]
workflow1: {"name": "myWorkflow", "type": "topicModelingPipeline",
            "schedulerType": "Slurm", "options": {"configFilePath":
            "CS_abstracts.py", "spotPrice": "1.57",
            "s3BucketName": "mys3bucket", "useCCQ": "true",
            "requestedInstanceTypes": "x1.16xlarge"}}

# Environment Template definition
[noSharedFilesystem]
description: Creates a CloudyCluster Environment that contains a
single Slurm Scheduler, a Login, and a NAT instance. It does not
contain any shared filesystems.
vpcCidr: 10.0.0.0/16
scheduler1: {"type": "Slurm", "ccq": "true",
             "instanceType": "t2.small", "name": "mySlurm"}
login1: {"name": "Login", "instanceType": "t2.small"}
nat1: {"volumeType": "SSD", "instanceType": "t2.micro",
       "accessFrom": "0.0.0.0/0"}
```

Fig. 2. Sample PAW configuration file, including an integrated workflow and CloudyCluster environment template.

the `Computation` and `Environment Templates` sections are optional. A sample configuration file that illustrates each of these sections is shown in Fig. 2.

The `UserInfo` section contains the information about the users that will run the jobs/workflows specified in the configuration file. In the case of the CloudyCluster implementation, the specified users are created within CloudyCluster and provisioned to the created environment. The `General` section contains parameters about PAW's configuration such as the cloud type and the environment name.

The `Cloud Type Settings` section is named dynamically and the name is based on the environment and cloud type being used. This section contains the information relevant to the desired environment and cloud types. For example, in this implementation of PAW the only EnvironmentType implemented is CloudyCluster and the only CloudType implemented is AWS, so the section header is titled `CloudyClusterAws`, as shown in Fig. 2. This section contains the information required to set up CloudyCluster on AWS, such as the region, keypair, VPC, and subnet. Much of this information is already pre-defined for the user. However, there are some parameters, such as the VPC ID and Subnet ID, that are unique to each user and do not have default values.

The `Environment Settings` section is also dynamically named, and contains parameters that pertain specifically to the type of environment being created. The sample configuration file section header shows `CloudyClusterEnvironment`. The environment key pair, region, and cluster configuration required by CloudyCluster are defined in this section.

The `Computation` section is an optional section of the configuration file. This section allows users to define job scripts or user-defined workflows that they want to execute on the newly created environment. These user-defined workflows, such as the topic modeling workflow used for our experiments, can be created by the user and integrated into PAW directly, which minimizes the configuration required. When extended to multiple environments, PAW allows users to define a single workflow that can be submitted to multiple newly created environments with a single click. Any job script, local or remote, that the user wants to run on a cluster environment can also be defined in this section. PAW submits the specified script to the environment and executes it using the scheduler defined by the environment.

The `Environment Templates` section is also optional and dynamically named. This section of the configuration file is utilized by the CloudyCluster Environment template generator to generate CloudyCluster Environment templates that can then be referenced in the `Environment Settings` section to reduce the length of future configuration files. In the sample configuration file shown in Fig. 2, we define the `noSharedFilesystem` template. The parameters in this section define the CloudyCluster environment configuration that will be put into the template. PAW comes with a number of pre-generated templates for researchers to use and extend as customized environment templates. The template generator supports the full range of customization options provided by CloudyCluster which allows researchers to fine tune their environments to better suit the requirements of their research.

*2) Resources:* Within PAW, the Resource classes interface directly with the underlying cloud or architecture provisioner. PAW's Resource base class defines three methods: *createControlResources*, *monitorControlResources*, and *deleteControlResources*. Adding a new underlying cloud or architecture involves adding a class that implements these methods. These methods are utilized by PAW to create the Control Resources that are required for interfacing with the cluster environment creation tools. For the initial implementation of PAW, the AWS Resource class has been defined to create the Control Resources required to utilize CloudyCluster within AWS. However, some clouds or architectures may not require the creation of any Control Resources, in which case this component is optional.

In PAW, the *createControlResources* method performs the actions required to create the resources needed to launch a new environment. The *monitorControlResources* function allows PAW to detect when the creation of the new resources created by the *createControlResources* method have completed successfully. *deleteControlResources* provides the commands to delete the resources previously created by the *createControlResources* method from the underlying cloud.

*3) Environments and Environment Templates:* The Environment classes interface with the chosen environment creation tool to create, monitor, and delete environments of that

particular type. A cluster environment defined within PAW contains all of the computational resources required to execute the workflows or jobs specified by the user. An environment consists of a combination of a scheduler, a shared filesystem, a NAT instance, login and compute nodes (i.e., instances), and a parallel filesystem. The components within an environment can be customized to fit the user's needs through the use of the included environment template generator.

The current environment template generator is built to generate templates for CloudyCluster environments, although support for other environment types can be added in the future. To create a template, the user defines a section in a configuration file with the required configuration and runs the template generator script with the configuration file. The environment template generator also includes the ability to manage and list the templates that have already been created. There are a number of pre-generated templates and configuration files included with PAW. The environment parameters utilized by the template generator are specified in human readable terms that do not require extensive knowledge of CloudyCluster.

*4) Schedulers:* The Scheduler classes within PAW allow it to interact with the different HPC schedulers within the created environments. Each of these classes provides an interface into a scheduler that allows PAW to submit, monitor, delete, and modify jobs within the scheduler. Adding a new scheduler is done by adding a new scheduler class that implements the Scheduler base class. For the current iteration of PAW, the CCQ scheduler is the only scheduler that is fully implemented. By utilizing CCQ, PAW is able to communicate using CCQ's web based APIs instead of having to make command line requests directly to the schedulers, which makes obtaining information from the scheduler easier and more efficient. CCQ already supports both the Torque and Slurm HPC schedulers, allowing PAW to communicate with these schedulers from a single interface. Supporting more schedulers such as OpenLava, Condor, and SGE is a topic of future work.

*5) Workflow Templates:* Workflow templates are a powerful feature of PAW that allow for easy customization of the system to specific domains and tasks. A workflow template in PAW is defined as a custom set of tasks or actions that are defined by the user that can then be submitted to an HPC scheduler to perform work. These workflow templates allow users to create and share their own workflows with other PAW users by simply sharing their custom Workflow classes.

A custom Workflow class must implement two methods, *run* and *monitor*. Code to generate or read a batch script file is added to the *run* method and any special monitoring code required for monitoring the completion of the workflow is put in the *monitor* function. For example, the *monitor* could include monitoring a specific storage location for an output file or monitoring the number of jobs. Users can also implement a custom workflow that utilizes another workflow manager such as SWIFT or QDO to execute the workflow. In these types of workflows PAW can be utilized to dynamically create the resources required by the workflow management tool.

We have implemented a Topic Modeling Pipeline Workflow class that reads in a configuration file, obtains the experiment parameters, performs a number of pre-processing tasks, and then submits a very large number of MPI jobs to the environment automatically. Since this workflow is integrated into PAW, we can run it by adding an extra line to the `Computation` section of the PAW configuration file as shown in Fig. 2

### B. PAW Operational Stages

There are five different operational stages within PAW: Create Control, Create Environment, Run Jobs/Workflows, Delete Environment, and Delete Control. These stages can be run together or separately, depending on the user's needs. When run together, PAW provides a "push button, get science" type of interface, as the user does not need to enter any information other than what is defined in the configuration file. All of the created resource IDs and required parameters are automatically passed from stage to stage. However, if the stages are not run consecutively or are run in chunks, the user will have to specify some parameters manually. Each stage is described in more detail in this section.

*1) Create Control Resources:* The first stage of PAW is the creation of the Control Resources. In this implementation of PAW, this stage creates the CloudyCluster Control Instance within AWS. In the context of a different environment type, the Control Resources stage can be defined as any resources required in order to allow PAW to interact with the chosen resource provisioning tools.

When using the CloudyCluster environment, PAW takes a CloudFormation template and launches a CloudyCluster Control Instance utilizing that CloudFormation template. It deploys and monitors the CloudFormation stack until the required resources have been launched successfully in AWS before continuing.

After the resources are ready, PAW performs the setup operations that are required to utilize CloudyCluster and CCQ. It creates a new CloudyCluster user, generates a new CloudyCluster App key, sets the Database capacity, obtains an authenticated web session with the CloudyCluster Control Instance, and makes sure the DNS name associated with the new Control Instance has propagated successfully. Once these tasks have completed, PAW moves to the Create Environment stage.

*2) Create Environment:* In the Create Environment stage, the cluster environment is created from the environment template specified in the PAW configuration file. The environment template contains the components of the environment and informs the Control Resources what it needs to create. PAW reads the template, transforms the template into the proper format required by the Control Resources and sends it to the Control Resources to begin the creation process. The environment creation can take a few minutes in AWS, depending on the size and scale of the environment specified. Once all of the resources specified in the environment template have been created successfully, PAW moves to the next stage, running the specified workflows and jobs.

*3) Run Workflow and Jobs:* During this stage of PAW the jobs are submitted to the HPC scheduler and any workflows that are specified in the configuration file are executed. This is done through utilizing the Scheduler classes previously described. For the initial PAW implementation, the submission of jobs and workflows to the HPC scheduler is performed by CCQ. CCQ is a meta-scheduler that adds the ability for job based autoscaling, submission, and monitoring to supported HPC schedulers without any code modifications. PAW utilizes CCQ's web based APIs in order to securely communicate with the CloudyCluster environment and perform the requested actions. If the job/workflow does not require all of the compute resources to be provisioned before executing, CCQ also provides the capability to allow the HPC scheduler to begin running the job/workflow before all of the compute resources have been launched.

For a singular job script there is no extra configuration needed. In this case, the user specifies within the configuration file the path to the job script, if the job script should be uploaded, and what remote directory they want the job to run in. CCQ uploads the job script if required, and then submits the job to the HPC scheduler. This allows researchers with extremely minimal experience with the commercial cloud to run batch job scripts on the commercial cloud with no extra configuration.

For a user-implemented workflow, PAW runs the code in the *run* method. Upon completion, PAW sends the job script to the scheduler, which executes the job script. After submission, the code in the *monitoring* section is executed, which allows PAW to determine when the workflow/job has completed so that the de-provisioning of the resources can begin.

In the initial version of PAW, when a job script or workflow is submitted to the CCQ scheduler the compute resources required for the operation of the workflow/job are dynamically created utilizing CCQ. This means that the user does not have to specify the resources required for computation ahead of time, or need to pay for these resources when they are not needed. Instead, PAW allows for the dynamic provisioning of the compute resources at workflow/job runtime, giving researchers more control over their costs and more flexibility in choosing the types of resources to utilize. By provisioning compute resources in this fashion, PAW allows researchers to provision specific resources for specific workflows/jobs quickly and efficiently. This eliminates the need for researchers to utilize the same instance type for every job/workflow and allows them to change the resources they execute the workflow on to meet different execution needs.

The handling of incomplete tasks and failed jobs is left to the user. PAW will notify the user if any of the workflows/jobs fail to complete, but will not re-submit the failed workflows/jobs. The reason for this decision is twofold. First, it may be the case that failed jobs may be able to be ignored, as is the case with many Monte Carlo type simulations. Secondly, PAW allows for the use of different workflow tools to be integrated into the custom workflows, and many of these tools have their own error handling and job re-submission routines.

*4) Delete Environment:* Even when the workflows/jobs complete, users still pay for the resources that have been created until they are deleted. PAW offers the ability to automatically delete and terminate all the created resources after the job/workflow has completed. This is an optional step and may not be desired in all cases. If specified, PAW will begin to terminate all of the created resources once it has determined that the workflows/jobs are done. This includes the shared filesystems and any information that was stored within the environment. If this stage is utilized, it is important that the workflows/jobs upload their results to a persistent storage location, such as S3 or another cloud storage service, so that data produced by the jobs/workflows is still available after the environment has been deleted.

If there is no environment to be deleted, such as when PAW is running on a local cluster environment, then this stage can be safely skipped and is not required to run PAW. Once the environment has been deleted successfully, PAW moves to the next stage, deleting the control resources.

*5) Delete Control Resources:* If the environment being utilized requires the creation of Control Resources, PAW can automatically delete them after the environment has been successfully deleted. Deleting the Control Resources means that any logs that may have been generated about the environment creation/deletion will be deleted and any access to the previous environment or Control Resources will be removed. These logs can be manually copied to a persistent storage location if desired. This is the final cleanup stage of PAW. Once the Control Resources have been deleted, there will be no resources created by PAW running within the resource provider chosen, and there will be no further costs to the user other than the costs of output and log files that remain in persistent storage.

## V. Case Study

To demonstrate and validate the scalability and flexibility of PAW we utilize an integrated topic modeling workflow that performs high throughput parameter sweeps. Topic modeling is a common text analysis technique in machine learning [17], [18]. In our experiment we run Parallel Latent Dirichlet Allocation (PLDA) [19] and vary numerous experimental parameters (e.g., the number of topics) to better understand how they affect the output models. We also run each experimental parameter combination many times in order to capture its tolerance to random seeds.

```
ARCH_NUM_NODES=5000
ARCH_NUM_CORES=30
ARCH_WALLTIME="24:00:00"
ARCH_MEM=16
DATASET_NAME="NIPS"
METHOD_NUM_NODES=1
METHOD_NUM_CORES=30
METHOD="PLDA"
METHOD_TOPICS="10 11 12 13 14 15 16 17 18 19 20"
METHOD_ITERS="1000 1500 2000 2500 3000"
METHOD_BURN_IN="500"
REPETITIONS=100
```

Fig. 3. Sample PLDA experiment descriptor file.

To set up this workflow, a user writes a human readable experiment descriptor file, such as the one in Figure 3. This file contains basic information required by the workflow, such as the number of compute nodes (i.e., instances) required, as well as method specific information such as which dataset to use and what experimental parameters to test. Once PAW has created the necessary resources, it submits an initial batch job that retrieves the required dependencies and executes a Python script. This script expands the experiment descriptor file into its component jobs, which are then submitted directly to the HPC scheduler. PAW monitors the progress of these jobs, and deletes the compute nodes once the jobs have completed. Each of these jobs runs PLDA on a particular experimental parameter combination on the specified dataset.

We conducted our scalability experiments on two datasets: the full text NIPS conference proceedings [20], and a set of abstracts from computer science publications provided to us by Elsevier. These experiments ranged in size from 278 compute instances up to 5,000 compute instances. Figure 4 shows the results of these experiments. Graphs A, B, C, and D show results from the CS abstracts experiment. These graphs are truncated on the right for space reasons. Graphs E and F show the results from the NIPS experiment, and contain the complete execution of PAW workflow resources.

For each of the experiments, all of the compute instances were created dynamically by PAW as specified by the parameters in the workflow configuration. Graph A shows a timeline of the number of each AWS instance type that was created. For this experiment we chose to utilize two different instance types, c4.2xlarge and c4.xlarge, because they fit the requirements for our workflow and are not too expensive. When submitted through PAW, we were able to provision 2,778 c4.xlarge instances and 2,222 c4.2xlarge instances in about 25 minutes. Graph B shows a timeline of the total vCPUs that were provisioned during the experiment. A vCPU is defined as a hyperthread of an Intel Xeon core [21]. As graph B shows, during the CS abstract experiment our 5,000 instances had a total of 28,832 vCPUs available for computation.

Graph C shows a timeline of the number of instances provisioned and the number of batch jobs submitted by the topic modeling workflow. As mentioned, in the topic modeling workflow the first batch job that is submitted generates and submits the rest of the batch jobs for the experiment. This job is submitted and begins running within 5 minutes of the submission of the workflow to PAW. The initial job finishes generating the experiments and submitting them to the HPC scheduler about 8 minutes after submission. As shown in graph C, the number of pending jobs within the environment rises quickly to almost 2,000 and the number of running jobs starts to increase. As more instances start to register with the HPC scheduler, more jobs start to run until all jobs are running. The timeline of the relationship between the number of running instances and the number of instances registered with the HPC scheduler is shown in Graph D. Graph D illustrates how quickly the instances go from the initial provision step to fully running and ready to be assigned computation. The first instance registers with the HPC scheduler within 5 minutes of the submission of the workflow to PAW and that 95% of the instances register within 20 minutes.

Graphs E and F show the results of the NIPS experiment. This experiment illustrates PAW's ability to not only create the computational resources but also to delete the computational resources after the workflow has completed. Graph E shows a timeline of the total number of instances provisioned during this experiment. Graph F shows a timeline of the number of running instances along with the total pending and running jobs in the environment. Graph F shows that first job is submitted about 5 minutes after the workflow was submitted to PAW and that the 3,000 jobs were generated and submitted within 10 minutes. The 278 instances provisioned by PAW completed 3,210 jobs in about 30 minutes. Then, as shown on the right side of graph F, PAW detected that the workflow had completed and, within two minutes, terminated all of the provisioned instances. This quick detection of job completion and deletion of resources is critical for managing costs when running on commercial cloud environments.

An issue to consider when running an HPC workflow within a commercial cloud is cost, especially for researchers with access to traditional resources. The cost varies with the type of instances and other cloud resources, and the execution time of the workflow. For our experiments we utilized the AWS Spot Market so as to obtain the lowest cost possible. The estimated cost to operate the 5,000 instance cluster environment using our maximum Spot bid price with more than 28,000 vCPUs in AWS is $777.80 per hour, or about $0.028 per vCPU hour. However, depending on Spot Market variations the costs can often be less. Whether this cost is justified is a business decision, and depends on the possible reduced time to solution when running in the commercial cloud, performance characteristics of the workload, and the availability or scheduling contention of traditional resources.

## VI. Conclusions and Future Work

We have presented an automated, dynamic cluster Provisioning And Workflow (PAW) management tool for parallel scientific applications in the cloud. This paper has described the core architecture concepts of PAW, the modularity of PAW, and the steps for adding a user defined workflow to PAW. We have described the initial implementation of PAW utilizing CloudyCluster on AWS as well as the different stages of PAW and the execution of the example topic modeling workflow.

By utilizing PAW, researchers are able to run custom defined parallel scientific workflows within AWS just as they would on an HPC cluster, and without any knowledge of how AWS works. Researchers have exclusive access to the environment and the HPC scheduler, which allows them to use computing resources without competition with other researchers.

The integrated topic modeling pipeline workflow demonstrates the scalability of PAW. With the current implementation using CloudyCluster and AWS, PAW provisioned a workflow utilizing 5,000 instances and more than 28,000 cores in 25

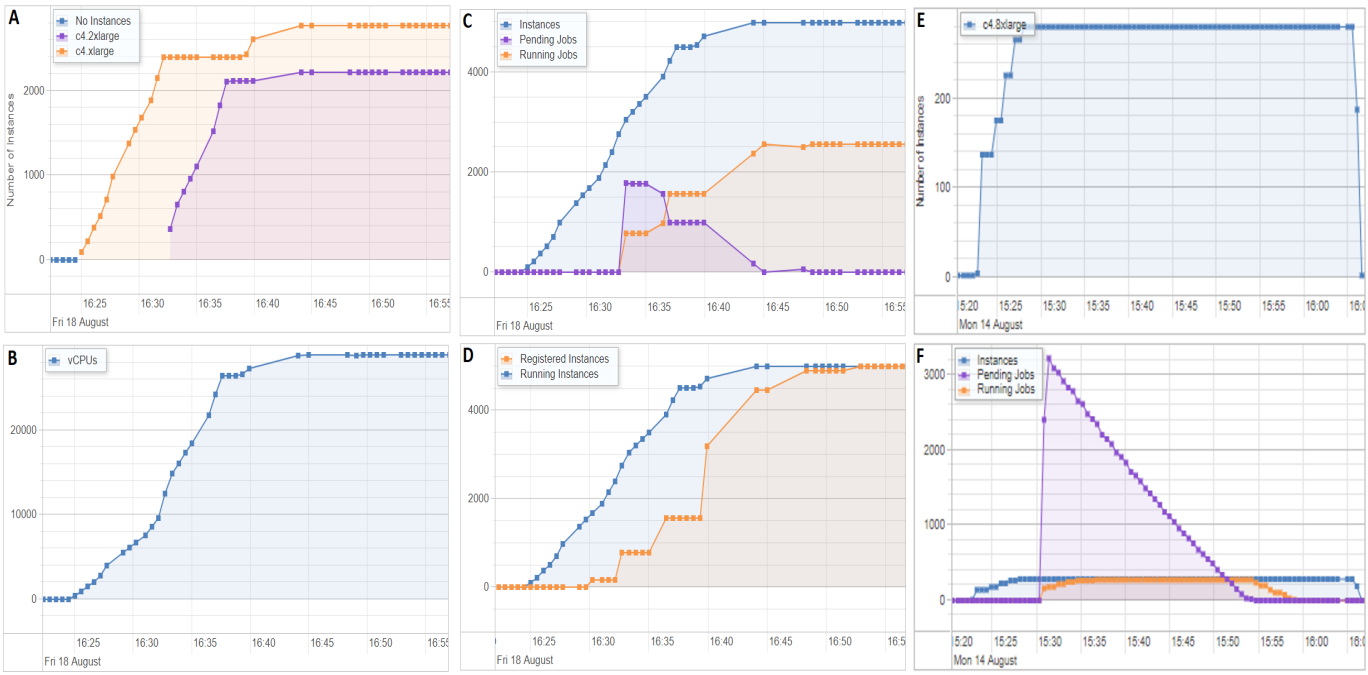


Fig. 4. Experimental runs of the topic modeling workflow utilizing PAW. **A)** A timeline of the number of each instance type launched during the CS Abstracts experiment. **B)** A timeline of the total number of vCPUs provisioned across all instance types for the CS Abstracts experiment. **C)** A timeline of the total number of instances, pending and running jobs during the CS Abstracts experiment. **D)** A timeline of the running instances and the number of instances registered with the HPC scheduler during the CS Abstracts experiment. **E)** A timeline of the total number of instances running during the NIPS experiment. **F)** A timeline of the number of running instances, pending jobs, and running jobs during the NIPS experiment.

minutes. The first job of the workflow started less than 10 minutes after the initial launch of the workflow to PAW. The other jobs were submitted and began executing as the resources registered with the scheduler and became available.

There are many features that we would like to add to PAW. A key goal is to extend PAW to different cluster environments and cloud providers. PAW has the flexibility to provision resources and manage workflows for multiple types of environments both static and dynamic. In this way, PAW can enable workflows to run on multiple environments without any refactoring. Complementary goals are to refine the process of creating and implementing custom workflows within PAW, and to improve the process of creating, implementing, and sharing workflows among researchers. The PAW source code is available for download at https://www.cs.clemson.edu/dice/.

## Acknowledgements

We acknowledge support from AWS, Omnibond, HPCC Systems, LexisNexis Risk Solutions, RELX Group, Elsevier Scopus, and GAANN award #P200A150310.

## References

[1] "Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017." *Gartner*, www.gartner.com/newsroom/id/3616417. Accessed 4 Sept. 2017.

[2] Bottum, Jim, et al. "The Future of Cloud for Academic Research Computing.", Apr. 2017, DOI: 10.13140/RG.2.2.10071.88489. 2017.

[3] "Tigres: Template Interfaces for Agile Parallel Data-Intensive Science." Tigres, tigres.lbl.gov/home. Accessed 3 Sept. 2017.

[4] "Is FireWorks for me?" Introduction to FireWorks (Workflow software) FireWorks 1.5.2 documentation, materialsproject.github.io/fireworks/index.html. Accessed 3 Sept. 2017.

[5] "QDO." *QDO Overview*, bitbucket.org/berkeleylab/qdo. Accessed 4 Sept. 2017.

[6] "A simple tool for fast, easy scripting on big machines." *The Swift Parallel Scripting Language*, swift-lang.org/main/. Accessed 4 Sept. 2017.

[7] Vahi, Karan, et al. "Pegasus WMS." *Pegasus WMS*, 24 Aug. 2017, pegasus.isi.edu/. Accessed 4 Sept. 2017.

[8] "DAGMan." *HTCondor - Directed Acyclic Graph Manager*, research.cs.wisc.edu/htcondor/dagman/dagman.html. Accessed 4 Sept. 2017.

[9] "AWS Batch Product Details." *Amazon Web Services, Inc.*, aws.amazon.com/batch/details/. Accessed 3 Sept. 2017.

[10] "Galaxy CloudMan." *Galaxy Community Hub*, galaxyproject.org/cloudman/. Accessed 3 Sept. 2017.

[11] "ElasticHPC." *ElasticHPC*, www.elastichpc.org/. Accessed 3 Sept. 2017.

[12] "HTCondor User Manual." *HTCondor Version 8.7.2 Manual*, research.cs.wisc.edu/htcondor/manual/v8.7. Accessed 3 Sept. 2017

[13] "Amazon Web Services (AWS) - Cloud Computing Services." *Amazon Web Services, Inc.*, aws.amazon.com/. Accessed 3 Sept. 2017.

[14] "AWS Cloud Credits for Research FAQ." *Amazon Web Services, Inc.*, aws.amazon.com/research-credits/faq/. Accessed 6 Sept. 2017.

[15] Posey, Brandon M. "Dynamic HPC Clusters within Amazon Web Services (AWS)." *Clemson University*, 2016, tigerprints.clemson.edu/all_theses/2392. Accessed 3 Sept. 2017.

[16] "Self Service HPC In The Cloud." *CloudyCluster*, www.cloudycluster.com/. Accessed 3 Sept. 2017.

[17] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." *Journal of machine Learning research* 3.Jan (2003): 993-1022.

[18] Blei, David M. "Probabilistic topic models." *Communications of the ACM* 55.4 (2012): 77-84.

[19] Wang, Yi, et al. "PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications." *AAIM* 9 (2009): 301-314.

[20] Amir Globerson, et al. "Euclidean Embedding of Co-occurrence Data." *JMLR* 8, 2007.

[21] "Amazon EC2 Instance Types Amazon Web Services (AWS)." *Amazon Web Services, Inc.*, aws.amazon.com/ec2/instance-types/. Accessed 3 Sept. 2017.