Pegasus in the Cloud: Science Automation through Workflow Technologies

Ewa Deelman, Karan Vahi, Mats Rynge, Gideon Juve, Rajiv Mayani, Rafael Ferreira da Silva
University of Southern California Information Sciences Institute
Contact author: Ewa Deelman, deelman@isi.edu

Abstract

The Pegasus Workflow Management System maps abstract, resource-independent workflow descriptions onto distributed computing resources. As a result of this planning process, Pegasus workflows are portable across different infrastructures, can be optimized for performance and efficiency, and can be automatically mapped to many different storage systems and data flows. This approach makes Pegasus a powerful solution for executing scientific workflows in the cloud.

1. Introduction

Scientific workflows are being used to orchestrate large-scale computations in a wide variety of scientific domains, including astronomy, bioinformatics, earthquake science, physics, and many others. Workflows provide automation of simulation and data analysis pipelines, ensuring that the computations are executed reliably, efficiently and correctly. The use of workflow technologies brings with it a number of benefits over traditional approaches such as shell scripting, including:

1. Ease of computation expression: Workflows provide high-level interfaces that rely on simple file formats, workflow-specific programming languages, and visual composition tools.
2. Ease of reuse/repurposing: As a result of the declarative nature of the workflow specification, portions or the entire workflow can be reused by others or modified to fit new requirements.
3. Failure management: Each step in a workflow can be automatically retried to deal with faulty hardware and transient failures, and the entire workflow can be checkpointed and resumed.
4. Parallel computing: Workflows enable computations and data to be processed in parallel on distributed resources.
5. Provenance tracking: As the workflow is executing, information about the data processed, the computations invoked, and the resources used is saved and can be inspected when verifying and validating the results.

Workflow applications execute in a variety of environments, including laptops, campus clusters, clouds, such as Chameleon or Amazon Web Services, distributed high-throughput resources, such as those managed by HTCondor, and high-performance computing (HPC) clusters, such as those operated by the XSEDE collaboration and the Department of Energy's national labs.

Different applications, based on their computational needs, rely on different computing resources and on different aspects of scientific workflows. For example bioinformatics applications that perform genomic processing may execute on clusters "close" to the sequencing machine. Additionally, there is a growing number of bioinformatics software packages and workflows that the community wants to share and reuse.

Astronomy workflows, such as science-grade image mosaicking, can be used by individual researchers to find new celestial objects, or by collaborations to generate large data resources that are useful to the entire astronomy community. Individual researchers might run these workflows on their desktop or laptop, while large data resources might be generated on on commercial clouds, which provide quick turnaround time to single-core workloads. Finally, there are applications such as those in earthquake science that need high-performance computing resources to execute parallel jobs within a workflow.

2. Pegasus Workflow Management System

The Pegasus Workflow Management System (Pegasus WMS) [1] enables users to describe their computations as workflows in a high-level language that is agnostic of the physical resources where the workflow is executed. This abstract description of the user's computation can be mapped to many different execution environments, ranging from laptops to HPC clusters.

Figure 1 provides a high-level schematic illustrating how scientists interface with Pegasus and the target execution environments. The workflow system is installed on a workflow *submit node* from where users submit their workflows. The submit node is usually decoupled from the distributed computational resources on which workflow jobs are executed. The workflow system is responsible for submitting jobs to remote resources, for managing the associated job and data lifecycle, and for providing status updates to users. This approach supports the "submit locally/compute globally" paradigm, where computations are managed from a central location, but executed on distributed resources.

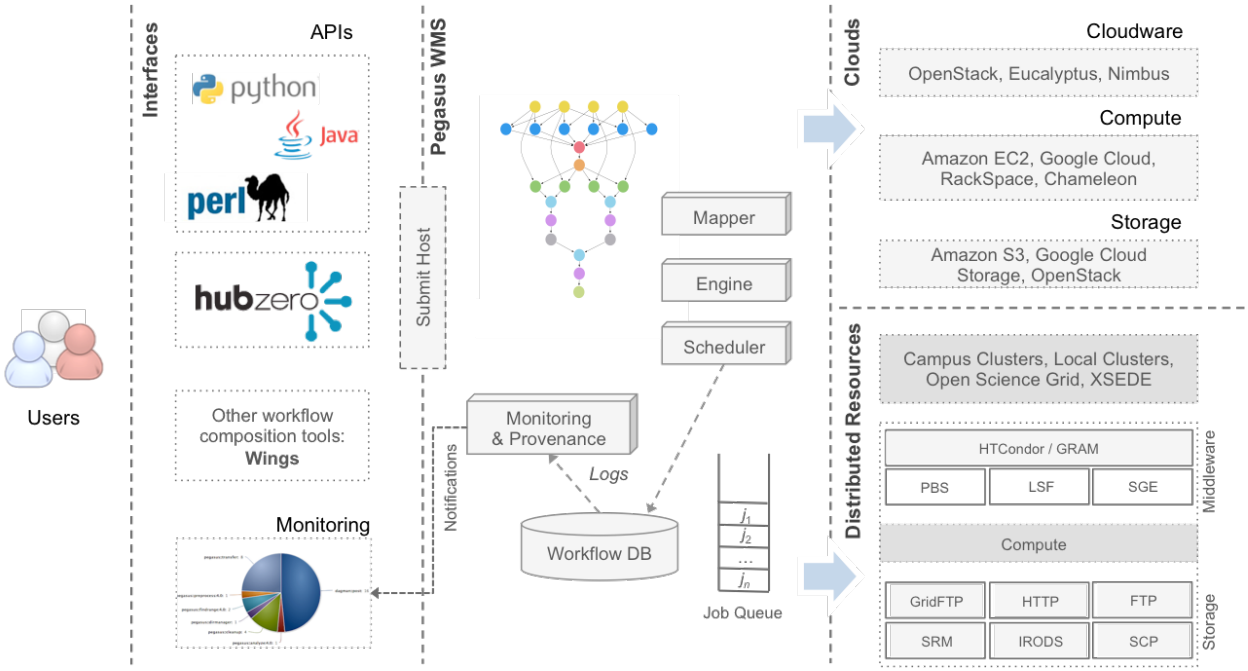

Figure 1: Pegasus WMS Architecture.

The Pegasus system is composed of four main components:

1. Pegasus Mapper: Generates an executable workflow based on an abstract workflow description provided by the user or workflow composition system. It finds the appropriate software, data, and computational resources required for workflow execution. The Mapper can also restructure the workflow to optimize performance and add jobs for data management and provenance information generation.

2. Workflow Engine (HTCondor DAGMan): Executes the tasks defined by the workflow in order of their dependencies. DAGMan determines when jobs are ready to run and submits them to the HTCondor queue for execution. It also retries failed jobs and checkpoints failed workflows.

3. Job Scheduler (HTCondor Schedd): Maintains a queue of ready jobs, and manages their execution on local and remote resources.

4. Workflow Monitor: Updates the workflow database with runtime provenance and performance information, notifies users of important events such as job failures, and provides a web interface for users to monitor and debug their workflows.

Workflows are described using an abstract format—called the DAX—that is independent of the target execution environment. This format enables portability, sharing and collaboration, and allows users to focus on designing and describing their pipelines without worrying about how the workflows will be executed. This approach enables Pegasus to move workflows between different execution environments and optimize workflows for each environment. The optimization takes into account how the data is moved and accessed while executing on these sites, and how computations should be managed for maximum efficiency and performance. Our users also benefit from the ability to execute workflows in newer execution environments as improvements are made to the system. Migration between or across systems does not require changes to the abstract workflow description.

The DAX captures, at a high-level, the jobs that make up the user's computation, the input and output datasets each job references, and the dependencies between the jobs that determine the order in which they can be executed. Figure 2 illustrates the DAX for a two-node hello world workflow and a schematic of the abstract to executable workflow mapping done by Pegasus.

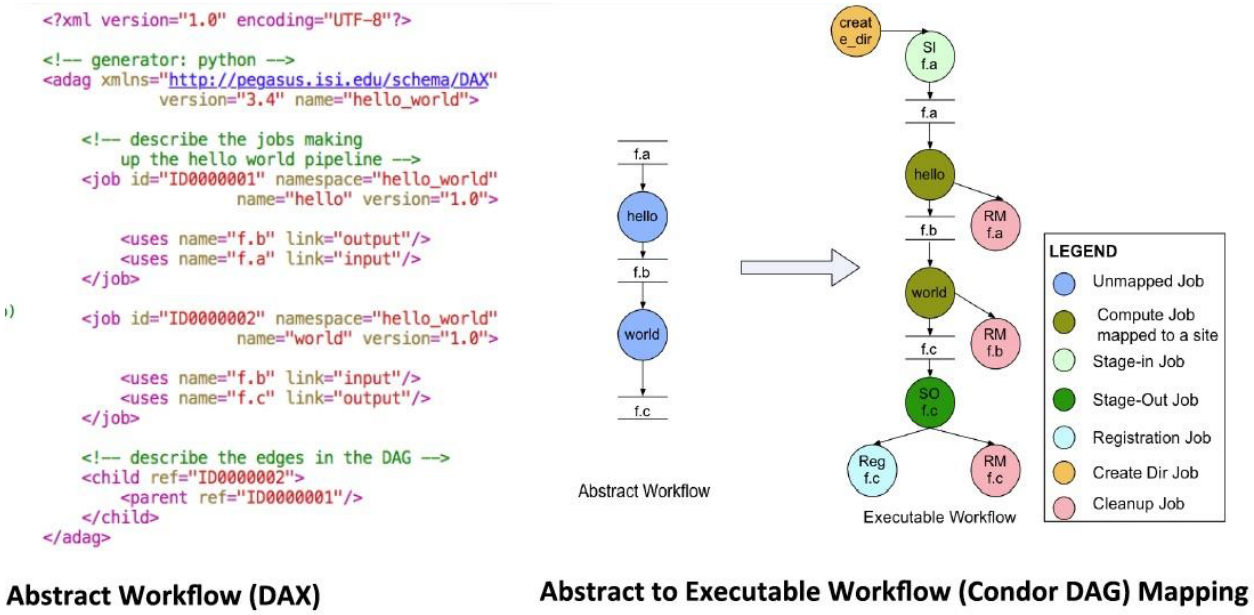

Figure 2: Workflow Design and Mapping

When creating the abstract workflow, users refer to executables and files by logical identifiers instead of physical paths or URLs. The Pegasus Mapper uses various information catalogs (data replica locations, executable locations, available computing resources) to resolve these logical identifiers and generate an executable workflow for a specific target environment. The Mapper adds auxiliary jobs that are responsible for transferring input and output data, for removing datasets that are no longer required, and for registering data in information catalogs for future discovery and reuse.

During the mapping process Pegasus performs optimizations including:

- **Data Movement** - Selection of the appropriate input file replicas and data transfer mechanism to use. For example, if an input file already exists at the resource, then the Mapper will setup the executable workflow to use that file instead of staging a file from an external location. Different data transfer mechanisms are employed depending on where the input, intermediate and output files are located and placed. For example, data movement jobs can transfer input files hosted on a user's FTP server to Amazon S3, where they can be used by compute jobs executing in Amazon EC2, and then transfer the outputs from EC2 or S3 back to the user's FTP server.
- **Data Reuse** - If during the data discovery phase, the Pegasus Mapper finds that a subset of the workflow's outputs already exists, the workflow is automatically pruned to avoid recomputing the existing outputs. This data reuse feature is commonly used for projects with overlapping datasets and workflows, and also enables failure recovery where users can use the partially computed results of a previously failed run. Data registration jobs in the executable workflow add information about generated outputs to an information catalog for future data discovery and reuse.
- **Data Cleanup** - Cleanup tasks are added to the executable workflow in order to reduce the storage footprint of the workflow at runtime.

- **Job Clustering** - The mapper can cluster multiple tasks from the DAX into larger jobs for performance and scalability. For example, short running tasks can be clustered to minimize job scheduling overheads.

In any distributed system, failures are a common occurrence, and pose a significant challenge for users to debug. The Pegasus system automatically records the state of the executable workflow in a database at runtime and provides a variety of debugging and monitoring tools that allow users to easily detect and troubleshoot failures in their workflows. The system also captures and records performance information such as workflow and job runtimes, execution hosts and their characteristics, memory usage, etc. This information can be used to build models for performance prediction of applications.

3. Running Pegasus Workflows on Cloud Infrastructures

There are many advantages to using clouds for scientific workflows, including: on-demand resource provisioning, the ability to allocate resources that match the needs of the workflow, such as large memory instances, and the ability to use custom software configurations that support applications with complex dependencies. In addition, users can easily share their computing environments with colleagues and reviewers, thus increasing collaboration and the reproducibility of scientific results.

Pegasus workflows can easily be deployed in the cloud by configuring cloud instances as an HTCondor pool. The VM image used for worker instances in the pool should contain HTCondor, Pegasus, and the application, and should be configured to contact the submit node to receive jobs. The submit node can be deployed inside or outside the cloud, depending on the needs and preferences of the user. For data storage, the pool can be configured with a shared file system such as NFS, or Pegasus can plan the workflow to use an object store such as Amazon S3 or transfer data from the submit node for each job.

In the early days of cloud infrastructures, Pegasus encountered some common issues in trying to adjust to this new model. Provisioning and image management were especially difficult, as Pegasus users were accustomed to running workflows on professionally managed infrastructures such as HPC clusters. With clouds, users were suddenly responsible for more of the software stack. Not only did the user have to think about managing their computation, but also the underlying system components such as the OS, libraries, schedulers, and, in some cases, storage systems. The maturing of the cloud and development of higher-level tooling helped. Within the Pegasus project, the Wrangler[3] system was designed to setup virtual clusters with instance dependencies, and the PRECIP[4] framework was created to manage repeatable cloud based computer science experiments.

The ability to dynamically scale the pool up (provision more instances) and down (kill instances) based on number of jobs in the queue are important considerations as they impact the runtime and the cost of the workflows. HTCondor is designed to be deployed on resources not owned by the pool administrator, such as across administrative domains of a university campus, and across wide area networks. It also uses soft state and has automatic job failover and recovery to enable resources to be added or removed on the fly. These features enable cloud autoscaling of HTCondor pools and, by extension, Pegasus workflows. We have used tools such as Amazon Auto Scaling and Scalr to automatically scale pools based on demand.

Upscaling decisions are usually based on either the load of existing compute instances, or the number of idle jobs in the queue on the submit node. Downscaling can be more difficult since there is a possibility of killing running jobs. This problem is more acute when running instances have multiple cores and are configured to run multiple jobs at the same time. One solution is to only downscale instances that don't have any jobs assigned to them. This solution works fine for workflows with jobs of relatively short duration. For workflows with a few long-running jobs, a more sophisticated downscaling policy might be required to consolidate the jobs onto as few instances as possible, and terminate instances that have only one job running. Any jobs that get killed will be rescheduled automatically onto another instance. Another approach is to signal the HTCondor daemons on the compute instances to stop accepting new jobs, and let the currently running jobs finish before terminating the instance. While individual pieces (HTCondor configuration recipes) exist that enable downscaling, more research and development is required to develop a software component that is able to automatically inspect the local job queue, and implement user defined policies for downscaling.

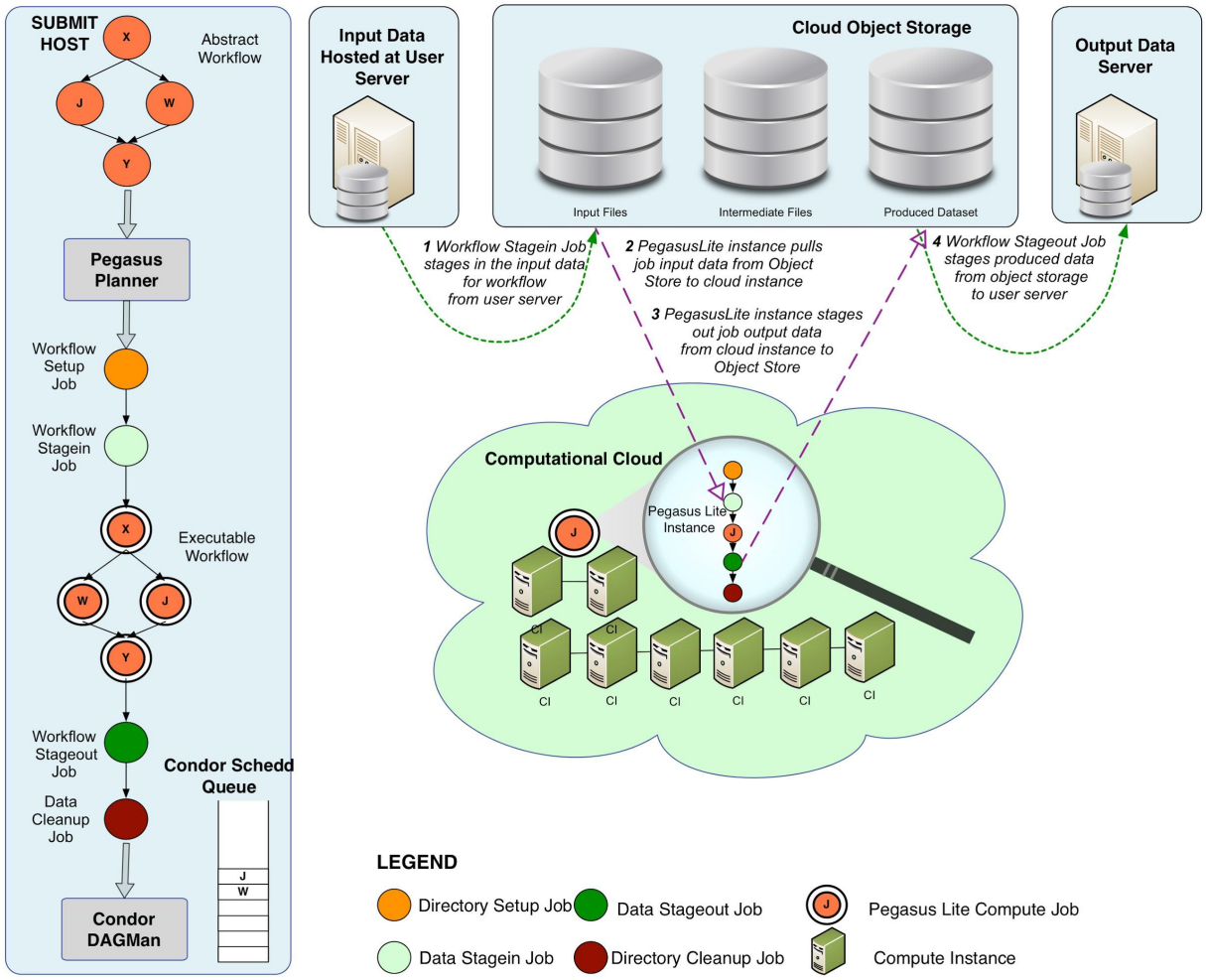

Figure 3: Data Flow for Pegasus Workflows in Cloud Environments - the magnifying glass highlights the data management performed by the Pegasus added data/provenance layer around the user task.

Resource provisioning and job management are obviously important to be able to execute workflows in a cloud environment, but the major cloud-enabling change to Pegasus was in data management. Before clouds, Pegasus only had one data management approach, which was based on the HPC cluster model. In that approach, Pegasus uses the shared file system for exchanging data between jobs, and jobs interact with data directly on the shared file system. With the advent of clouds, Pegasus' data management was updated to make use of object stores. Pegasus already tracks all the files in a workflow, so the change was focused on how to move data around and how to continue to offer POSIX file system access to jobs, which was important as most scientific codes require a basic file system to do I/O. Our approach is based on using a local disk on the VM instance for job execution. Pegasus ensures that whenever a user task is executed, it is launched on a POSIX file system in a directory where the inputs required by the task are already present. In the new data management approach, Pegasus automatically wraps user jobs with tools to manage data in the executable workflow. First, files required by a task are fetched from an object store and put on local file system of the instance, then the job is executed, and, finally, any output files are transferred back to the object store. The object store does not need to be co-located with the compute instances, and could be outside the cloud or even hosted at another cloud provider. This separation enables workflow executions to span a mix of resources, for example when using multiple cloud providers at the same time, or mixing cloud and non-cloud resources [2]. Figure 3 illustrates a typical data flow for Pegasus workflows in cloud environments.

An additional advantage that clouds offer is the ability to package up the compute environment along with application codes. An increasing number of scientific workflow users are developing virtual machine images that have Pegasus configured to start automatically when the virtual machine starts up. These virtual appliances provide an application-specific web interface that enables other scientists in the community to upload their input datasets, or refer to existing datasets in the cloud, launch their scientific analysis, and monitor them.

4. Example Applications

The Galactic Plane[5] project used cloud workflows to generate image mosaics for astronomy. The goal of the project was to take a set of existing data products from different telescopes and different product specifications, and process the data in such a way that the produced datasets would look like it came from a single instrument. In collaboration with NASA's Infrared Processing and Analysis Center (IPAC), Amazon Web Services, and the Pegasus team, an ensemble of 16 massive hierarchical workflows were executed on Amazon AWS. EC2 was used for the computation and S3 for both the intermediate files, as well as the final output storage. The 16 hierarchical workflows in the ensemble each had 1,001 sub-workflows with an average of 6 million tasks each. Input data access were over publically accessible data find interfaces, also used by other projects. In order to not overwhelm these interfaces, the number of compute instances were limited by using manual provisioning with the AWS web interface. The final data product of the project was an image atlas containing large-scale mosaics of the galactic plane in 16 wavelengths, totaling 45 TBs.

The rSEQ[6] project packaged GT-FAR, a transcriptome optional RNA-seq analysis pipeline, along with Pegasus as a virtual appliance to bring accurate expression analysis to researchers without significant computational resources. The appliance provides an application specific web interface that enables users

to upload custom datasets, start and monitor the progress of their GT-FAR analysis, receive notifications when the analysis is done, and makes the output datasets available for download from S3. It generates error reports based on information reported by the Pegasus system that can be used to identify application errors. The appliance also leverages Pegasus Data Reuse capabilities to bypass the computationally expensive index generation phase at the start of the pipeline, by tracking and storing the index files when they are first generated.

5. Conclusion

In this article we provided an overview of how scientists can model their computational pipelines as workflows using the Pegasus Workflow Management System and the advantages that this system provides. While Pegasus was initially developed for a more traditional HPC environment, its unique planning and optimization approach has enabled users to take advantage of cloud resources without having to change their workflow descriptions. The overall system design allows Pegasus users to fully take advantage of cloud concepts such as auto scaling, object stores and image sharing.

Acknowledgements

This work was supported by the National Science Foundation under grants number ACI-1148515 and FutureGrid 0910812.

References

[1] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny and Kent Wenger, Pegasus: a Workflow Management System for Science Automation, Future Generation Computer Systems, 46, pp. 17-35, 2015.

[2] Gideon Juve, Mats Rynge, Ewa Deelman, Jens-S. Vöckler and G. Bruce Berriman, Comparing FutureGrid, Amazon EC2, and Open Science Grid for Scientific Workflows, Computing in Science and Engineering, 15:4, pp. 20-29, 2013.

[3] Gideon Juve and Ewa Deelman, Wrangler: Virtual Cluster Provisioning for the Cloud, Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC11), 2011.

[4] Sepideh Azarnoosh, Mats Rynge, Gideon Juve, Ewa Deelman Michal Nieć, Maciej Malawski and Rafael Ferreira da Silva, Introducing PRECIP: An API for Managing Repeatable Experiments in the Cloud, Workshop on Cloud Computing for Research Collaborations (CRC), 2013

[5] Mats Rynge, Gideon Juve, Jamie Kinney, John Good, G. Bruce Berriman, Ann Merrihew and Ewa Deelman, Producing an Infrared Multiwavelength Galactic Plane Atlas using Montage, Pegasus and Amazon Web Services, 23rd Annual Astronomical Data Analysis Software and Systems (ADASS) Conference, 2013.

[6] Ying Wang, Gaurang Mehta, Rajiv Mayani, Jingxi Lu, Tade Souaiaia, Yangho Chen, Andrew Clark, Hee Jae Yoon, Lin Wan, Oleg V. Evgrafov, James A. Knowles, Ewa Deelman and Ting Chen, RseqFlow: Workflows for RNA-Seq Data Analysis, Bioinformatics, 27:18, pp. 2598-2600, 2011.