Automating Environmental Computing Applications with Scientific Workflows

Rafael Ferreira da Silva*, Ewa Deelman*, Rosa Filgueira‡, Karan Vahi*, Mats Rynge*
Rajiv Mayani*, Benjamin Mayer§
* University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA
‡ School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK
§ Oak Ridge National Laboratory, Oak Ridge, TN, USA
{rafsilva,deelman,vahi,rynge,mayani}@isi.edu, rosa.filgueira@ed.ac.uk, mayerbw@ornl.gov

***Abstract*—Computational environmental science applications have evolved and become more complex over the last decade. In order to cope with the needs of such applications, computational methods and technologies have emerged to support the execution of these applications on heterogeneous, distributed systems. Among them are workflow management systems such as Pegasus. Pegasus is being used by researchers to model seismic wave propagation, to discover new celestial objects, to study RNA critical to human brain development, and to investigate other important research questions. This paper provides an introduction to scientific workflows and describes Pegasus and its main features. The paper highlights how the environmental science community has used Pegasus to automate their scientific workflow executions on high performance and high throughput computing systems by presenting three use cases: two Earth science workflows, and a climate science workflow.**

***Index Terms*—Environmental Computing; Scientific workflows; Pegasus Workflow Management System**

I. Introduction

Scientific computing is the mainstream method for scientists that want to extract the maximum information out of their data—which are often obtained from scientific instruments such as the USArray Transportable Array [1] (seismic instruments). These computations often require the processing and analysis of vast amounts of data to understand complex system behaviors and interactions. In order to support the computational and data needs of today's science, scientists have used distributed computing infrastructures (e.g., grids, clouds, campus clusters, and supercomputers) to efficiently compute their analyses in a reliable and scalable way. This is the case in fields such as seismology, climate and ocean modeling, hydrology, astronomy, bioinformatics, and physics.

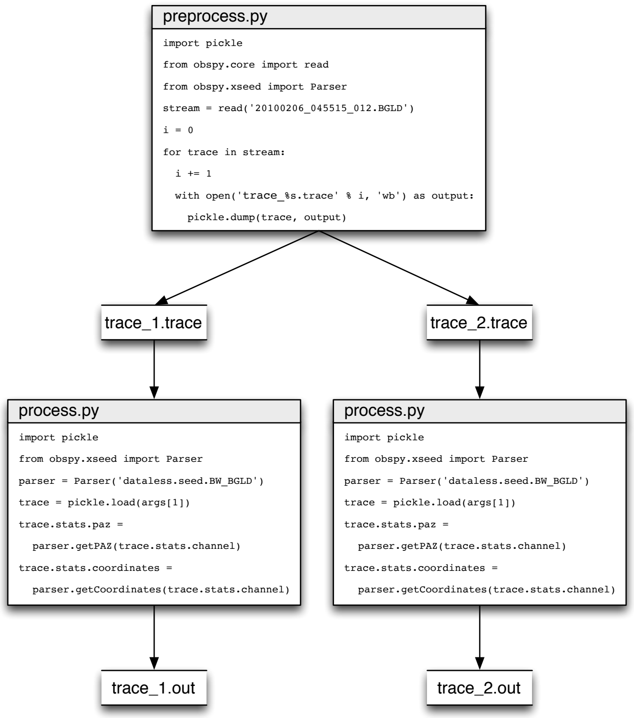
In the meantime, scientific workflows have emerged as a flexible representation to declaratively express complex applications with data and control dependencies. As a result, many scientific workflow management systems (WMSs) have been developed [2]–[5], and they have been intensively used by various research communities [6]. Workflows allow researchers to easily express multi-step computational pipelines involving simulation, data analysis, and visualization; and to abstract the specificities of the computing environment allowing scientists to focus on the experiment definition and the analysis rather than the configuration and deployment of the execution environment. Additional advantages of workflows include reusability (aids reproducibility), data provenance, fault-tolerance, parallel distributed computations, among others.

In this paper, we provide an overview of scientific workflows, presenting their main characteristics and applicability, as well as examples of workflow management systems (WMSs) that address different classes of problems. We describe the Pegasus WMS [2], a well-established workflow system that is widely used by the scientific community to seamlessly manage the execution of computations across distributed, heterogeneous systems. Then, we illustrate three use cases from the Earth science (CyberShake [7] and Seismic Ambient Noise Cross-Correlation [3]) and the climate science (ACME [8]) domains, highlighting how these communities have benefited from the use of scientific workflows, in particular Pegasus, for running small- and large-scale computations in the cloud and on national cyberinfrastructures.

This paper is structured as follows. Section II describes previous works, and highlights selected applications. Section III presents an overview of scientific workflows and workflow management systems. Section IV describes Pegasus, its properties, and its main capabilities. Section V describes the three environmental computing workflows for the Earth science and climate science domains, and Section VI concludes the paper.

II. Related Work

Scientific workflows have been used for over a decade, and a plethora of workflow management systems have been developed to attend the different needs of computational science applications [2]–[5]. As a result, workflow management systems became the focus of many characterization studies and surveys [9], [10]. In environmental computational science, most of the applications that use workflows target large-scale analyses on distributed systems [3], [7], [8]. Workflows are also extremely useful to manage the execution of dispersed analyses on heterogeneous systems [11]. Therefore, in this paper we provide an introduction to scientific workflows highlighting their main properties and functionalities, and how they may fit the requirements of environmental computing applications.



```
import pickle
from obspy.core import read
from obspy.xseed import Parser
stream = read('20100206_045515_012.BGLD')
i = 0
for trace in stream:
  i += 1
  with open('trace_%s.trace' % i, 'wb') as output:
    pickle.dump(trace, output)
```

```
import pickle
from obspy.xseed import Parser
parser = Parser('dataless.seed.BW_BGLD')
trace = pickle.load(args[1])
trace.stats.paz =
  parser.getPAZ(trace.stats.channel)
trace.stats.coordinates =
  parser.getCoordinates(trace.stats.channel)
```

```
import pickle
from obspy.xseed import Parser
parser = Parser('dataless.seed.BW_BGLD')
trace = pickle.load(args[1])
trace.stats.paz =
  parser.getPAZ(trace.stats.channel)
trace.stats.coordinates =
  parser.getCoordinates(trace.stats.channel)
```

Fig. 1: An example of a workflow that processes seismology data. The workflow is composed of two task types (`preprocess.py` and `process.py`), which are Python programs.

III. Scientific Workflows

Scientific workflows are an important abstraction for the composition and automation of complex scientific applications in a broad range of domains. They provide an abstraction above the individual application components and often abstract the details of the execution environment. Workflows allow scientists to easily express multi-step computational tasks, for example retrieve data from an instrument or a database, reformat the data, and run an analysis; while automating data movement between workflow processing stages. Typically, a workflow is described as a directed acyclic graph (DAG), where the nodes are tasks and the edges are dependencies (often data dependency). Workflows may also contain conditional statements (`if-then-else` statements, exception handling) or loops. Fig. 1 shows an example of a workflow composed of two tasks (`preprocess.py` and `process.py`), where dependencies are expressed as data files. In this example, the `process.py` tasks only start their execution, once their input data (`trace_1.trace` and `trace_2.trace`) are available, i.e. the task `preprocess.py` is successfully completed.

Task. In scientific workflows, a task often represents a program (or a script) written in any programming language, which is seen as a black box by the workflow engine—the workflow system is not aware of the intrinsics of the application code, it is limited to the invocation of the application with given parameters. Few workflow management systems require the implementation of the workflow mechanism as part of the application code (white box application). This is typically the case of shared memory applications. A task may also represent a local or remote Web service, sub-workflows, etc. In addition, a single task execution may require a single CPU (embarrassingly parallel, no communication between processes), or multiple CPUs (e.g., MPI-like, i.e. tightly coupled processes—there is communication between processes).

Dependency. Workflow dependencies define the order that tasks should be executed in a workflow—the outcome of parent tasks are the input for child tasks. Typically, dependencies are based on flow of data between tasks (as shown in Fig. 1). However, they can as be expressed as any kind of event that could trigger the execution of the subsequent tasks or advance the workflow execution. Other types of dependencies include conditional statements, exceptions (error handling), user triggered action (the workflow waits for the user's input to proceed with the execution), among others.

A. *Workflow Composition*

Scientific workflows are described as high-level abstraction languages which mask the complexity of execution infrastructures to the scientist. The example shown in Fig. 1 describes the problem that the scientist is aiming to solve as a multi-step computational analysis. Note that this representation does not provide any detail about the underlying execution platform, i.e., it is an *abstract workflow*. During execution, the workflow system concretizes the abstract workflow by mapping it to the execution infrastructure, which therefore is now called the *executable workflow*. The mapping phase may add additional tasks to the workflow to enable its execution on the computing platform (e.g., stage in input data), and performs static workflow optimizations (e.g., data cleanup, task clustering, workflow reduction, etc.).

Besides the execution environment abstraction, abstract workflows also foster reproducibility through workflow reuse and data provenance. The abstract workflow can run in different execution environments with no alterations. For instance, scientists can share their workflows within the scientific community, and other scientists can reproduce an equivalent analysis following the same recipe. Data provenance (the process of tracing and recording the origins of data and its movement) allows scientists to easily trace the origin of the data produced.

Workflow Patterns and Design. The shape of a workflow may assume a variety of different forms. For instance, in neuroinformatics and bioinformatics, most of the workflows are structured as pipelines (sequential tasks, no parallelization). More complex workflows are composed of parallel splits and merges, decision points (conditions), etc. [12]. As a result, several workflow languages were developed to describe workflows. A workflow language is a formalism expressing the causal/temporal dependencies among a number of tasks to execute. Currently, there is no standard way to describe a workflow (most workflow systems defined their own language), although there are some efforts in trying to develop a common workflow language [13]. Current workflow management systems provide either a graphical interface to build workflows (drag-and-drop) or command line interfaces (via APIs), or both. While graphical interfaces provide easy mechanisms to create workflows, they may limit the versatility of workflows to handle large-scale complex applications—it is challenging to visualize and manipulate a graph with hundreds of thousands of nodes. On the other hand, command line interfaces require programming skills. However, they easily scale to tackle large-scale complex problems.

B. *Workflow Execution*

Workflow interpretation and execution are handled by a workflow engine that manages the execution of the application on the computational infrastructure. As aforementioned, several workflow management systems (WMSs) have been developed to address different requirements of diverse domain applications. For example, workflow systems differ in support of task-oriented or stream-based workflows. Task-oriented WMSs typically support parallel computations, where tasks may run at the same time and there is no dependency between them (e.g., Pegasus [2], Makeflow [4], etc.). On the other hand, stream-based WMSs provide concurrent task execution, i.e. a task (with precedence constraint) can start its execution before its parent tasks have completed (e.g, dispel4py [3], Nextflow [5], etc.).

Workflows may simply run in the local user's environment (e.g., laptop or desktop machine), or in large, diverse, heterogeneous, distributed systems. The scale of the system depends mostly on the complexity of the applications, and on the capabilities of the systems. For example, parameter sweep studies are often composed of independent small tasks that can significantly benefit from grid or cloud computing environments. Parallel applications often require specialized systems such as clusters and supercomputers (for large scale applications), which can provide high connectivity with low latency (required for the high frequency of message exchanges). Note that a workflow may be composed of different application types, where some of the workflow tasks are optimized to run in high throughput systems (e.g., grids), and the remainder require high performance computing platforms (e.g., supercomputers). The data movement between the different computing environment can then automatically handled by the WMS.

Fig. 2 shows an example of a workflow that runs in different computing infrastructures. The workflow is composed of a *split* job, which divides the input data into several small chunks of data that can be processed independently (e.g., data preparation, filtering, etc.). These tasks (`process.R`) can then run in high throughput systems. The following step (`merge.c`) is a parallel task (MPI application) that will merge the contents of the processed data (e.g., cross-correlation analysis), and requires high performance systems such as clusters or supercomputers. Finally, the outcome of the analysis may be visualized locally on the scientist's desktop machine.

IV. Pegasus Workflow Management System

In this section, we present Pegasus [2], a well established scientific workflow management system for automating, recovering, and debugging scientific computations. Pegasus focuses on scalable, reliable, and efficient workflow execution on a wide range of systems, from the user's desktop to state-of-the-art high-throughput and high-performance computing systems.

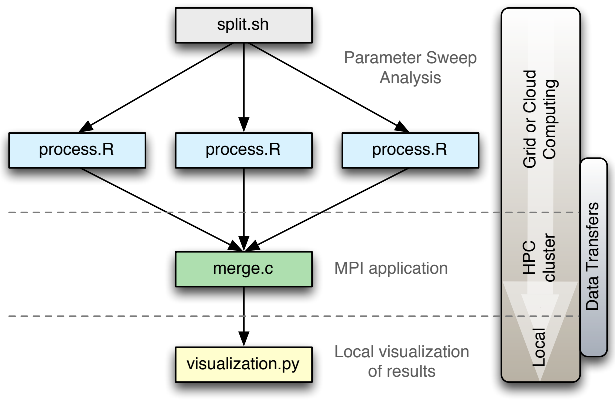


Fig. 2: An illustration of a workflow to that runs in a heterogeneous computing environment. The top part of the workflow performs a parameter sweep analysis and runs on grids or cloud environments. The middle part is an MPI-like application and requires HPC resources. The bottom part is a simple visualization script that runs locally in the scientist's computer. Data transfers between different computing environments are automatically handled by the workflow system.

Pegasus bridges the scientific domain and the execution environment by automatically mapping high-level abstract workflow descriptions onto distributed heterogeneous resources. It can manage data on behalf of the user, infer the required data transfers, register data into catalogs, and capture performance information while maintaining a common user interface for workflow submission. In Pegasus, workflows are described abstractly as directed acyclic graphs (DAGs), where nodes represent individual computational tasks and the edges represent data and control dependencies between tasks. The abstract workflows do not have any information regarding physical resources or physical locations of data and executables. The abstract workflow description is represented as a DAX (DAG in XML), describing all tasks, their dependencies, their required inputs, their expected outputs, and their invocation arguments.

A. *Overview of Pegasus Functionalities*

Pegasus has a number of features to facilitate and automate workflow executions, and foster collaboration and reproducible research:

1) *Versatility:* Pegasus defined workflows can run in different computing environments without modifications to the abstract workflow. The same workflow can run on a single system or across a heterogeneous set of resources.

2) *Data Management:* Pegasus automatically handles data transfers. It autonomously determines the best available replica of the input data; performs data movement between tasks (within the same computer system, or across different computing platforms); and records output data into data catalogs.
3) *Fault-tolerance:* Pegasus provides several mechanisms to mitigate faults. Tasks and data transfers are automatically retried in case of failures, alternative data sources are used to stage the data, etc. Automated storage constraints mechanisms ensure that storage limits are honored by removing unnecessary data during the workflow execution. Pegasus also provides workflow recovery support through workflow-level checkpointing, and by providing a *rescue* workflow, a reduced version of the original workflow containing only the tasks that were not executed.
4) *Performance Optimizations:* Pegasus performs automatic workflow optimizations for improving the efficiency of the workflow execution. Tasks are reordered, grouped, and prioritized in order to increase the overall workflow performance. Pegasus determines existing workflow's output data, and prunes the workflow to prevent unnecessary re-computation.
5) *Scalability:* Pegasus can run workflows composed of a few tasks up to millions of tasks. It also provides scaling capabilities to dynamically adapt to the number of available resources at runtime.
6) *Provenance:* Pegasus enacts reproducibility through its ability to systematically capture provenance information for the derived data products. In addition to data provenance, Pegasus captures task performance data, which aids debugging.

B. Creating Pegasus Workflows

Workflow composition within Pegasus is done via APIs. Pegasus provides application programming interfaces in Python, Java, R, and Perl to generate the abstract workflow as a directed acyclic graph in XML (DAX). Fig. 3 shows the pseudocode using the Python API for generating the seismology example workflow presented in Fig. 1. The advantages of using an API include the flexibility provided by programming languages for defining large-scale workflows. For instance, parameter sweep studies can be easily defined using a simple *for* loop; a variety of workflows can be generated based on conditional statements evaluated using the workflow's input data (i.e., *if* statements may define whether a set of tasks would be part of the workflow). A complete description of the APIs can be found in the Pegasus' documentation [14]. Note that the abstract workflow does not require any information about the files locations nor details about the computing platform.

Pegasus also provides mechanisms to define dynamic workflows, i.e., workflows that are generated at runtime. Dynamic workflows can be expressed as sub-workflows—a task of the main workflow is actually an invocation to a DAX generator, that will create a sub-workflow during execution based on the outcomes of the previous tasks (parent tasks).

```
#!/usr/bin/env python
from Pegasus.DAX3 import *

# Create a DAX
workflow = ADAG('seismology_workflow')

# Add a preprocess job
preprocess = Job('preprocess')
trace1 = File('trace_1.trace')
trace2 = File('trace_2.trace')
preprocess.uses(trace1, link=Link.OUTPUT)
preprocess.uses(trace2, link=Link.OUTPUT)
workflow.addJob(preprocess)

# Add left process jobs
process1 = Job('process')
out1 = File('trace_1.out')
process1.uses(trace1, link=Link.INPUT)
process1.uses(out1, link=Link.OUTPUT, transfer=True)
workflow.addJob(process1)

# Add right process jobs
process2 = Job('process')
out2 = File('trace_2.out')
process2.uses(trace2, link=Link.INPUT)
process2.uses(out2, link=Link.OUTPUT, transfer=True)
workflow.addJob(process2)

# Add dependencies
workflow.depends(parent=preprocess, child=process1)
workflow.depends(parent=preprocess, child=process2)
```

Fig. 3: Example of a DAX generator for the seismology example workflow shown in Fig. 1.

C. Executing Pegasus Workflows

For running workflows, Pegasus requires the specification of three different types of information catalogs: the (1) *site catalog*—describes the computing sites where the workflow tasks can be executed; the (2) *transformation catalog*—describes all of the executables (programs, also called *transformations*) used by the workflow tasks; and the (3) *replica catalog*—describes all of the input data stored on external servers. Pegasus uses the information provided in the catalogs to automatically map the abstract workflow onto an executable workflow (with all additional tasks to perform data stage in/out and optimizations).

Execution Environments. The simplest execution mode is the *local* method, where the workflow runs entirely in the scientist's machine. This method is particularly useful for development tests, or for scientists who want to automate small-scale analyses. Large-scale computations may be executed on grid computing platforms (e.g., Open Science Grid [15]), academic (e.g., NSF Chameleon [16]) and commercial (e.g., Amazon EC2 [17], Google Cloud [18], etc.) cloud computing platforms [19], campus clusters and national cyberinfrastructures (e.g., XSEDE [20]). During execution, the workflow is orchestrated to release tasks as they become available (parent tasks are completed), and data transfers are automatically performed within the same platform or across computing infrastructures. Note that the complexity of running workflows may increase depending on the selected infrastructure. For

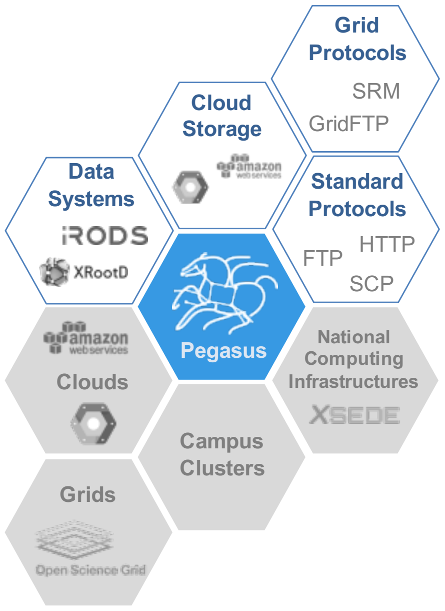


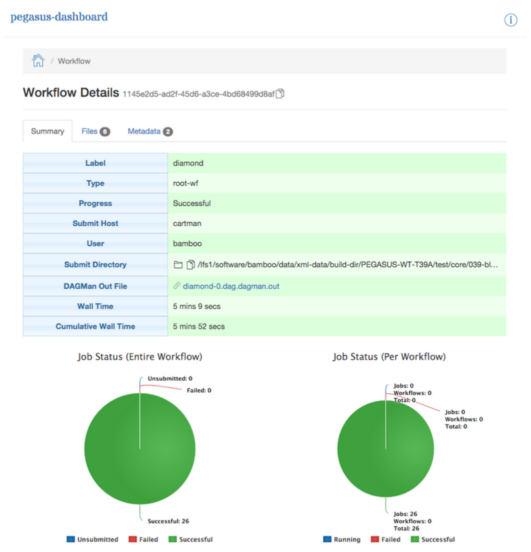
Fig. 4: Overview of execution environments (bottom grey hexagons), and data management protocols and services (top white hexagons) supported by Pegasus.

example, high-performance computing systems (HPC) often have stricter access, thus it may require additional steps to setup the workflow execution environment. The workflow execution complexity may also be impacted by the number of different heterogeneous computational platforms the workflow runs on. On the other hande, this may significantly improve the efficiency of the workflow execution.

Data Management. During the mapping process, Pegasus performs several optimizations as highlighted above. It provides mechanisms to automate data transfers from a number of different data protocols and services. Pegasus handles from simple files copies (using the basic linux command `cp` or `symlink`) within the same resource, up to remote data transfers using standard protocols such as HTTP, FTP, and SCP. For workflow runs conducted in grid environments, Pegasus uses SRM and GridFTP protocols, the most used transfer protocols available in these systems. For runs in cloud environments, Pegasus also interacts with cloud storages such as Amazon S3 and Google Cloud Storage using their data transfer clients. Additionally, Pegasus also provides mechanisms to interact with advanced data system such as iRODS [21] and XRootD [22].

Fig. 4 shows an overview of all execution environments and data management services and protocols supported by the Pegasus system. Note that a workflow may be composed of several different environments, and data may be staged in/out from/to different protocols or services.

Monitoring. Workflow monitoring is fundamental for debugging and following the progress of the workflow execution. Pegasus provides a suite of tools for monitoring and debugging workflows at runtime or when the workflow execution is completed. The Pegasus Dashboard is a web-based application that provides real-time monitoring of workflow executions. It



Workflow Details 1145e2d5-ad2f-45d6-a3ce-4bd68499d8af

Label	diamond
Type	root-wf
Progress	Successful
Submit Host	cartman
User	bamboo
Submit Directory	/lfs1/software/bamboo/data/xml-data/build-dir/PEGASUS-WT-T39A/test/core/039-bl...
DAGMan Out File	diamond-0.dag.dagman.out
Wall Time	5 mins 9 secs
Cumulative Wall Time	5 mins 52 secs

Fig. 5: Screenshot of the Pegasus monitoring dashboard.

shows the status of the workflow and its tasks, task characteristics, statistics, and performance metrics (Fig. 5). In addition to the dashboard, Pegasus provides command-line tools (e.g., `pegasus-statistics`, `pegasus-analyzer`) to inquiry specific issues or characteristics of the workflow.

V. Use Cases: Environmental Computing Workflow Applications

In the past years, the environmental science community has used scientific workflows to address the simulation of complex phenomena such as climate changes and earthquake hazards. Below, we describe three workflow applications that have used Pegasus as a mean to seamlessly automate the execution of environmental computing workflows in distributed systems.

A. *Cybershake*

As part of its research program of earthquake system science, the Southern California Earthquake Center (SCEC) has developed CyberShake [7], a high-performance computing software platform that uses 3D waveform modeling to calculate physics-based probabilistic seismic hazard analysis (PSHA) estimates for populated areas of California. A CyberShake hazard curve computation can be divided into two phases. In the first phase, a 3D mesh of approximately 1.2 billion elements is constructed and populated with seismic velocity data. This mesh is then used in a pair of wave propagation simulations that calculates and outputs strain Green tensors (SGTs). The SGT simulations use parallel wave propagation codes and typically run on 4,000 processors. In the second phase, individual contributions from over 400,000 different earthquakes are calculated using the SGTs, then these hazard contributions are aggregated to determine the overall seismic hazard. These second phase calculations are loosely coupled, short-running serial tasks. To produce a hazard map for Southern California, over 100 million of these tasks must be executed. The extensive heterogeneous computational requirements and large numbers of high-throughput tasks necessitate a high degree of flexibility and automation; as a result, SCEC utilizes Pegasus workflows for execution [23].

Recently, SCEC has conducted a study to run 336 Cybershake workflows over 38 days on Blue Waters (NCSA) and Titan (OLCF). Pegasus managed the execution of the workflows, which consumed about 1.4 million node-hours, and automated the management and movement of 550 TB of data within the workflows, where 197 TB were transferred from Titan to Blue Waters, and 9.8 TB staged back to USC HPC (∼7M files) [24]. The scalability required to achieve the science goal underscores the need for Pegasus to automate the execution and data management of SCEC's large-scale workflows.

B. *Seismic Ambient Noise Cross-Correlation*

Recently, the Pegasus team and the dispel4py [3] (a stream-based workflow system) team have collaborated to enable automated processing of real-time seismic interferometry and earthquake *repeater* analysis using data collected from the IRIS database [25]. Such analyses require a large number of waveform cross-correlations, which is computationally intensive. The workflow (Seismic Ambient Noise Cross-Correlation) periodically reads data from the repository (about every hour), and performs waveform cross-correlations analyses through thousands of computational tasks. The workflow consists of two main phases:

- `Preprocess`—each continuous time series from a given seismic station (called a trace), is subject to a series of treatments. The processing of each trace is independent from other traces, making this phase embarrassingly parallel (complexity O(n), where n is the number of stations); and
- `Cross-Correlation`—pairs all of the stations and calculates the cross-correlation for each pair (complexity O(n^2)).

The workflow is implemented as a hybrid workflow approach to enable the execution of data-intensive stream-based workflow applications across different e-Infrastructures. The workflow execution is performed in heterogeneous computing platforms described as Docker containers, which can be deployed and executed in cloud computing environments. Fig. 6 shows an illustration of the *Seismic Ambient Noise Cross-Correlation* workflow implementation. A single (non-stream) run of the workflow requests an hour data from IRIS services (USArrayTA [1], data from 394 stations), and executes the `Preprocess` phase in about 8 minutes (using the OpenMPI cluster deployed in `Container 2`), while the `Cross-correlation` phase requires about 2 hours for processing (using the Apache Storm [26] cluster deployed in `Container 3`). A stream-based version of the workflow reads data from IRIS services every 2 hours.

The stream-based executions of the *Seismic Ambient Noise Cross-Correlation* workflow are managed by dispel4py, while the data movement between different execution infrastructures, and the coordination of the application execution are automatically managed by Pegasus. This approach enables the distributed allocation of stream-based workflows (represented

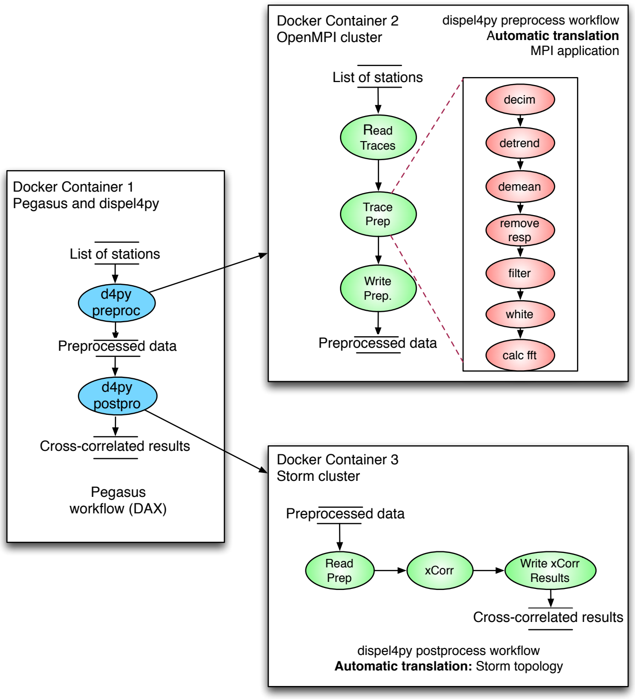


Fig. 6: Overview of the Seismic Ambient Noise Cross-Correlation workflow using Pegasus and dispel4py to automate computation on distributed resources (OpenMPI and Apache Storm clusters). Pegasus orchestrates the workflow execution, and manages data movement between different computing environments, while dispel4py executes stream-based workflows within a container.

as tasks of the Pegasus workflow) to the appropriate computing resources—allocate the workflow to the resource that could execute it in the most efficient way.

C. *ACME*

The Accelerated Climate Modeling for Energy (ACME) project is using coupled models of ocean, land, atmosphere and ice to study the complex interaction between climate change and societal energy requirements. The ACME workflow [8], [27] automates the manual effort involved in monitoring and resubmitting the model code in case of failures, and provides periodic reporting for validation of science outputs. The workflow divides a large climate simulation into several stages (Fig. 7). Each stage completes a portion of the total target simulation time. Each stage also produces history files, which are used by the workflow to automatically compute summary data called climatologies. This climatology data can be reviewed periodically by project scientists to ensure that the simulation is progressing as expected, so that problems can be identified, and corrections made, before computing resources are wasted. The workflow has been executed in high performance computing infrastructures due to its complex large-scale parallel computations. The number of nodes (or cores) needed is determined from the size of the grid problem. A simple workflow execution on Titan (OLCF) with four

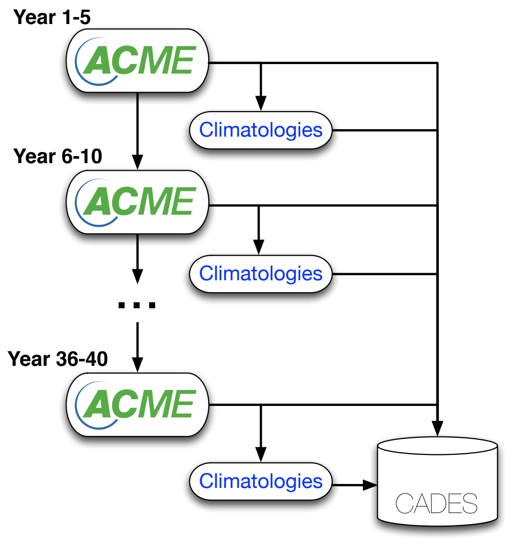


Fig. 7: Overview of the Accelerated Climate Modeling for Energy (ACME) workflow.

different parameter values for a number of simulation time units (5 years per stage), consumes over 50,000 CPU hours.

VI. Conclusion

In the past years, workflows have been used by several science domains to efficiently manage their computations on a number of different resources: from the scientist's desktop to complex, state-of-the-art systems. For instance, the environmental science community has used workflows to create earthquakes hazard maps, or to compute seismic noise cross-correlations. In this paper, we have presented an introduction to scientific workflows, emphasizing their main properties and how scientists may model their experiments as workflows. We described the Pegasus workflow management system, highlighting three use-cases where the environmental scientists have used Pegasus to advance their research methods.

Although workflows have been used by a small group of environmental scientists, we believe that more researchers can benefit from these technologies. The main goal of this paper is to provide practical guidance on how scientists, in particular environmental scientists, may benefit from the use of workflows. On the other hand, new workflow requirements such as in-situ analysis and visualization or real-time data stream analyses, also challenge the workflow research community to provide efficient and advanced tools and resources [28].

Acknowledgements

This work was funded by DOE under the contract number #DESC0012636, "Panorama–Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows"; and by the National Science Foundation under the SI^2–SSI program, award number 1148515.

References

[1] "USArray - Transportable Array," http://www.usarray.org/researchers/obs/transportable.

[2] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[3] R. Filgueira, A. Krause, M. Atkinson, I. Klampanos, and A. Moreno, "dispel4py: A python framework for data-intensive scientific computing," *International Journal of High Performance Computing Applications (IJHPCA)*, 2016.

[4] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. ACM, 2012, p. 1.

[5] "Nextflow," http://www.nextflow.io/index.html.

[6] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated, 2014.

[7] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner *et al.*, "Cybershake: A physics-based seismic hazard model for southern california," *Pure and Applied Geophysics*, vol. 168, no. 3-4, pp. 367–381, 2011.

[8] E. Deelman, C. Carothers, A. Mandal, B. Tierney, J. S. Vetter, I. Baldin, C. Castillo, G. Juve, D. Krol, V. Lynch, B. Mayer, J. Meredith, T. Proffen, P. Ruth, and R. Ferreira da Silva, "PANORAMA: An approach to performance modeling and diagnosis of extreme scale workflows," *International Journal of High Performance Computing Applications*, 2015.

[9] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "Hpctoolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[10] E. M. Bahsi, E. Ceyhan, and T. Kosar, "Conditional workflow management: A survey and analysis," *Scientific Programming*, vol. 15, no. 4, pp. 283–297, 2007.

[11] R. Filgueira, R. Ferreira da Silva, A. Krause, E. Deelman, and M. Atkinson, "Asterism: Pegasus and dispel4py hybrid workflows for data-intensive science," in *The Seventh International Workshop on Data-Intensive Computing in the Clouds (DataCloud), submitted*, 2016.

[12] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and parallel databases*, vol. 14, no. 1, pp. 5–51, 2003.

[13] "Common workflow language," http://www.commonwl.org.

[14] "Pegasus documentation," https://pegasus.isi.edu/documentation.

[15] "Open science grid," https://www.opensciencegrid.org.

[16] "Chameleon cloud," https://www.chameleoncloud.org.

[17] "Amazon elastic compute cloud," http://aws.amazon.com/ec2.

[18] "Google cloud platform," https://cloud.google.com.

[19] E. Deelman, K. Vahi, M. Rynge, G. Juve, R. Mayani, and R. Ferreira da Silva, "Pegasus in the cloud: Science automation through workflow technologies," *IEEE Internet Computing*, vol. 20, no. 1, pp. 70–76, 2016.

[20] "XSEDE," https://www.xsede.org.

[21] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert *et al.*, "iRODS primer: integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[22] L. Bauerdick, D. Benjamin, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, M. Ernst, R. Gardner, A. Hanushevsky, H. Ito *et al.*, "Using xrootd to federate regional storage," in *Journal of Physics: Conference Series*, vol. 396, no. 4. IOP Publishing, 2012, p. 042009.

[23] S. Callaghan, E. Deelman, D. Gunter, G. Juve, P. Maechling, C. Brooks, K. Vahi, K. Milner, R. Graves, E. Field *et al.*, "Scaling up workflow-based applications," *Journal of Computer and System Sciences*, vol. 76, no. 6, pp. 428–446, 2010.

[24] "SCEC Cybershake: Using Pegasus to run across OLCF Titan, Bluewaters and HPCC," https://pegasus.isi.edu/2016/02/18/pegasus-scec-2015/.

[25] "IRIS: Incorporated research institutions for seismology," https://www.iris.edu.

[26] "Apache storm," http://storm.apache.org.

[27] B. Mayer, P. Worley, R. Ferreira da Silva, and A. Gaddis, "Climate science performance, data and productivity on Titan," in *Cray User Group Conference*, 2015.

[28] R. Ferreira da Silva, W. Chen, G. Juve, K. Vahi, and E. Deelman, "Community resources for enabling and evaluating research on scientific workflows," in *10th IEEE International Conference on e-Science*, ser. eScience'14, 2014, pp. 177–184.