Asterism: Pegasus and dispel4py hybrid workflows for data-intensive science

Rosa Filgueira[1], Rafael Ferreira da Silva[2], Amrey Krause[3]
Ewa Deelman[2], Malcolm Atkinson[4]
[1]British Geological Survey, Lyell Centre, Edinburgh EH14 4AP
[2]University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA
[3]EPCC, University of Edinburgh, Edinburgh EH8 9LE, UK
[4]School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK
rosa@bgs.ac.uk, {rafsilva,deelman}@isi.edu, a.krause@epcc.ed.ac.uk,
malcolm.atkinson@ed.ac.uk

ABSTRACT

We present `Asterism`, an open source data-intensive framework, which combines the strengths of traditional workflow management systems with new parallel *stream-based* dataflow systems to run data-intensive applications across multiple heterogeneous resources, without users having to: re-formulate their methods according to different enactment engines; manage the data distribution across systems; parallelize their methods; co-place and schedule their methods with computing resources; and store and transfer large/small volumes of data. We also present the Data-Intensive workflows as a Service (`DIaaS`) model, which enables easy data-intensive workflow composition and deployment on clouds using containers. The feasibility of `Asterism` and `DIaaS` model have been evaluated using a real domain application on the NSF-Chameleon cloud. Experimental results shows how `Asterism` successfully and efficiently exploits combinations of diverse computational platforms, whereas `DIaaS` delivers specialized software to execute data-intensive applications in a scalable, efficient, and robust way reducing the engineering time and computational cost.

Keywords

Data-Intensive science, scientific workflows, stream-based system, deployment and reusability of execution environments

1. INTRODUCTION

In the era of Big Data Science, *research campaigns* are producing and consuming ever-growing data sets (e.g., models, raw data, etc.). These campaigns are considered data-intensive applications due to the large amount of data they analyze (or use), or due to the number of I/O operations involved in their executions. Data-Intensive applications need to be executable; that is, they need to (1) fetch their large set of input data from the wide area (e.g., data repositories); (2) apply methods between live (real-time produced data) and archived data applications (3) move data between stages when necessary; (4) perform computations (in a single computing resource or across several ones) for simulation, data preparation, analyses, and presentation; (5) clean-up the data that is not necessary for the execution; (6) store final results together with user-selected intermediates; and (7) provide a framework for diagnosis, validation, visualization, and the use of end-results [2].

DataCloud '16 November 14, 2016, Salt Lake City, UT, USA
© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Over the last years, scientific workflows have emerged as an important abstraction that allows scientists to easily model and express their data-intensive application with their dependencies. As a result, many scientific workflow management systems (WMSs) have been developed, and they have been intensively used by various research communities, e.g., astronomy, biology, computational engineering [28]. However, they usually address a set of the previous needs for modeling a data-intensive application. For example, *coarse-grained* composition workflow systems, such as `Pegasus` [8], are mainly mechanisms to organize and distribute computation regardless of the application language, or the target computing infrastructure. Alternatively, parallel dataflow systems (e.g., `dispel4py` [17]) have the ability to model the movement of data and methods as a series of connection (inputs and outputs), enabling concurrent pipelined and task parallelism, so tasks can execute as soon as the required resources are available. However, unlike *coarse-grained* WMSs, parallel datalfow systems often do not provide capacities to run workflows across e-Infrastructures.

In this paper, we present `Asterism`, a hybrid framework that provides facilities to run data-intensive *stream-based* applications across platforms on heterogeneous systems. `Asterism` provides simple and flexible high-level programming abstractions for coordination, data access, and data exchange. It blends the benefits of `dispel4py` with `Pegasus` systems:

- `Pegasus` is a mature, scalable, and robust *task-oriented* WMS, which offers facilities to run workflows on heterogeneous systems. These features have been demonstrated through wide number of applications and projects (e.g., LIGO gravitational wave detection analysis [27]);
- `dispel4py` is a new abstract parallel *stream-based* dataflow

system that has demonstrated its benefits by describing several applications during the VERCE [31] project. It offers mappings to several enactment engines used in HPC and data-intensive computing without users having to modify their codes, and it also smooths transitions from local development to scalable executions.

Therefore, the *stream-based* executions of an `Asterism` 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`.

Complementary to `Asterism`, we present the Data Intensive Workflow as a Service (`DIaaS`) model to provide scientists with a flexible and easy-to-use environment for running scientific applications within containers. In this model, all required software (workflow systems and execution engines) are packed into the containers, which significantly reduces the effort (and possible human errors) required by scientists or platform administrators to build such systems.
The rest of this paper is organized as follows. Section 2 provides background on scientific workflows, and the two workflow management systems used in this work, and highlights related work. Section 3 presents the `Asterism` framework, its functionalities, and main advantages. The experimental evaluation using Docker containers and a real workflow application are presented in Section 4. Section 5 describes the Data-Intensive workflow as a Service (`DIaaS`) model, and Section 6 summarizes our results and identifies future work.

2. BACKGROUND AND RELATED WORK

In this section, we introduce the main concepts, tools and relevant works, which are the foundations of this work.

2.1 Scientific workflows

Today's computational and data science applications process vast amounts of data for conducting large-scale simulations of underlying science phenomena. These applications may comprise thousands of computational tasks and process large datasets, which are often distributed and stored on heterogeneous resources. Scientific workflows have emerged as a flexible representation to declaratively express such complex applications with data and control dependencies, and have become mainstream in a number of different science domains [28]. In spite of impressive achievements to date, constructing workflows and orchestrating their executions on heterogeneous systems remain a fundamental challenge. As a result, many workflow management systems (WMSs) have been developed to fulfill specific requirements of different scientific communities [1,8,13,17,21,22,25,34]. Although these systems have a common goal, they often do not share all capabilities across different e-Infrastructures.

Typically, most WMSs support *task-oriented* workflows [1, 8, 13, 21, 22, 34], where the predominant model has stages that correspond to tasks, and the workflow organizes its enactment on a wide range of distributed computing infrastructures, normally arranging data transfer between stages via files. On the other hand, the growing volumes of scientific data, the increased focus on data-driven science and the achievable storage density doubling every 14 months (Kryder's Law [33]), severely stresses the available disk I/O – or more generally the bandwidth between RAM and external devices. This is driving increased adoption of *stream-based* [17, 25] for implementing data-intensive applications, as these avoid a write out to disk followed by reading in, or double that I/O load if files have to be moved. Significantly reducing the cost of data movement between stages makes it economic to compose very simple stages, e.g. format changes, which potentially intermingle with much more demanding stages. However, they are few *stream-based* systems that proive facilities to run applications across heterogenous systems, as we aim with `Asterism`. One of them, is the Storm extension presented in [24], which allows the generation of data stream processing applications that operate in a distributed Cloud environment. But this extension only enables to run Storm topologies across Clouds, whereas `Asterism` enables to map automatically `Asterism` workflows to Storm, MPI and Multiprocessing engines and across to a wide range of e-Infrastructures (e.g. Clouds, Clusters, Grid).

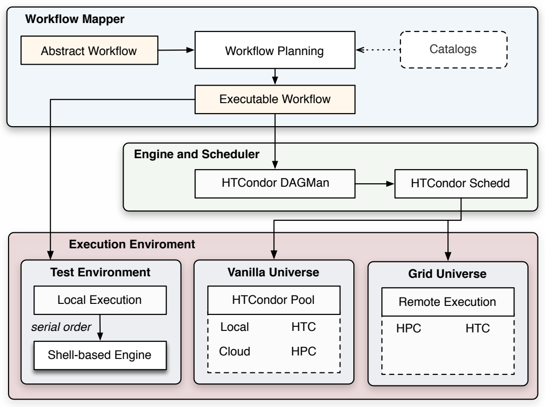


Figure 1: Overview of the `Pegasus` architecture.

2.2 Pegasus

`Pegasus` [8] 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. `Pegasus` bridges the scientific domain and the execution environment by automatically mapping high-level abstract workflows descriptions onto distributed resources. In `Pegasus`, workflows are described abstractly as directed acyclic graphs (DAGs), where the nodes are tasks and the edges are dependencies (often data dependency). The abstract workflow 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.

Figure 1 shows the main components of the `Pegasus` framework. The Workflow Mapper plans the workflow execution. During execution, `Pegasus` translates the abstract workflow into an executable workflow, determining the executables, data, and computational resources required for the execution. `Pegasus` maps executables to their installation paths or to a repository of stageable binaries defined in a catalog. Workflow execution with `Pegasus` includes data management, monitoring, and failure handling, and is managed by DAGMan [18]. Individual workflow tasks are managed by a task scheduler (HTCondor [29]), which supervises their execution on local and remote resources, including campus and national cyberinfrastructures, grids, and clouds.

Table 1: Main capabilities and paradigms of dispel4py and Pegasus workflow management systems.

Capabilities	*dispel4py*	*Pegasus*
Description	Abstract workflow language that maps at runtime to different enactment platforms and automatically parallelizes the workflow making use of the number of cores/processes available	Maps abstract workflows to executable workflows that can be executed by Condor DAGMan in heterogenous platforms. Performs optimizations for performance and reliability
Target	Enables scientists to focus on their scientific goals, avoiding distracting details and retaining flexibility over the computing infrastructure they use. Users only have to express their computational needs and connect them	Distributes compute-intensive workflows and handles automatic data transfer across computing resources. It can be seen as layer on top of DAGMan with capabilities for provenance, monitoring, and failure recovering
Nodes	Processing Elements (PEs), which are Python Objects	Jobs, which are executables (black box) of any type (bash scripts, python, C++, etc.)
Data transferred	Memory/stream	Files (I/O). It also supports remote I/O
External storage (data access)	Work in progress	Key element of Pegasus
Language	Python for describing PEs and their connections	DAX API in Python, Java, R, Perl
Parallelism	Automatic parallelism of PEs	Parallelism of nodes (jobs)
Application	Application modeled as a single workflow	Application modeled as N jobs
Distributed heterogeneous platforms	Not supported	N computing resources and types
Execution platforms	Sequential, MPI, Multiprocessing, Storm, Spark	Condor (DAGMan), PBS, SGE, etc.
Workflow scheduling	Workflow scheduled as one job	Static scheduling of workflow jobs
Conditional execution	Supports conditional execution of PEs	Not supported
Concurrent execution	Supports concurrent pipeline execution	Not supported
Failure recovering	Not supported	Supports failure recovering and check points at the workflow and application levels
Task/PE scheduling	Work in progress	Supports task scheduling and sub-workflows
Monitoring	Work in progress	Workflow- and task-level monitoring

2.3 dispel4py

The dispel4py system [17] is a parallel *stream-based* dataflow framework for formulating and executing data-intensive methods. It is based on a simple and abstract model of the logic required. That abstract model carefully eliminates details of target platforms and data-handling steps that can be automated. dispel4py is a Python implementation of the *Dispel* language to reach the extensive community of Python users. In contrast to *Dispel*, dispel4py not only constructs and describes abstract workflows, but encapsulates executables as Python objects. The model comprises nodes, called *Processing Elements* (PEs), connected through data streams. The PEs process units of data from each of their inputs, and emit units of data on each of their outputs. Each data stream carries a sequence of data units from its source, normally a PE's output port, to all of the input ports to which it is connected.

One of dispel4py's strengths is the ability to construct abstract workflows with no knowledge of the underlying execution infrastructure—this approach enables portability across different computing platforms without any migration cost imposed to users. Users can therefore focus on the design of their workflows, describing actions, input and output streams, and their connections. dispel4py then maps these descriptions to the enactment platforms. Currently, dispel4py provides mappings for Apache Storm [12], MPI [26], and Multiprocessing, as well as a Sequential mapping for development and small applications.

Table 1 highlights the main capabilities and differences between both, dispel4py and Pegasus, workflows systems.

2.4 Container Orchestration

Recently, Linux container technology has gained attention as it promises to transform the way software is shared, reused, developed, and deployed [19, 35]. The portability and ease of deployment makes Linux containers an ideal technology to be used in scientific workflow platforms. They are a form of virtualization that uses advanced kernel features, mainly *namespaces* and *cgroups*, to define different user spaces on top of a single kernel space. Docker [10] is a new but already very popular open source tool that combines: (1) performing Linux container (LXC) based operating system level virtualization, (2) portable deployment of containers across platforms, (3) component reuse, (4) sharing, (5) archiving, and (6) versioning of container images [6].

A particular advantage of Docker containers is that the resulting computational environment is immediately portable. Docker handles the packaging and execution of a container so that it works identically across different machines, while exposing the necessary interfaces for networking ports, volumes, and so forth, allowing other users to reconstruct the same computational environment. Additionally, Docker technology provides a convenient distribution service called *The Docker Hub* [11], which freely stores the pre-built images, along with their metadata, for download and reuse by others. Therefore, we have selected Docker containers to provide the Asterism framework with portability, easiness composition, and scalability across different e-Infrastructures.

Running workflows in cloud platforms have been the subject of several studies [7, 9, 22, 32], however they focus solely on a single workflow management system, and use emulation of a particular computer system (virtual machines).

3. ASTERISM: HYBRID WORKFLOWS FOR DATA-INTENSIVE APPLICATIONS

`Asterism` is a hybrid framework composed by `Pegasus` and `dispel4py` workflow systems. `Asterism` greatly simplifies the effort required to develop data-intensive applications that run across multiple heterogeneous resources distributed in the wide area, without its users having to (1) reformulate their methods according to different enactment engines; (2) manage the data-distribution across systems; (3) parallelize their methods; (4) co-place and schedule their methods with computing resources; and (5) store and transfer large/small volumes of data.

They key element of `Asterism` is that the selected WMSs complement each other's strengths (Table 1). On one hand, `dispel4py` allows developing scientific applications locally and then automatically parallelize and scale them on a wide range of e-infrastructures and enactment engines without any changes to the codes. On the other hand, `Pegasus` orchestrates the distributed execution of applications across e-Infrastructures offering capabilities such as portability, automated data management, recovery, debugging, and monitoring, without its users needing to worry about the particulars of the target execution systems.

We propose to represent the different parts that compose a complex multi-stage data-intensive application (e.g., preprocess, process, post-process, etc.) as `dispel4py` workflows, and use `Pegasus` to *connect* to orchestrate the distribution and execution of each `dispel4py` workflow (mapped as a task in `Pegasus`). The different levels of abstractions provided by both workflow management systems allow users to focus on the design of their applications at an abstract level, describing actions, input, and output streams; how the actions are connected; and in which computing resources the actions (or a set of them) should be executed. The *stream-based* execution is then managed by `dispel4py`, while the data movement between different execution platforms and the workflow engine (submit host) is managed by `Pegasus`. Figure 2 shows this process, in which an application is initially modeled with `dispel4py` by using three PEs. Later, these PEs are mapped into `Pegasus` tasks (*P1* and *P2* into task *T1*, and *P3* into task *T2*), representing then an `Asterism` workflow. `Asterism` users need to indicate for each `Pegasus` task which PEs should be wrapped in, which site each task should be executed, and with which `dispel4py` mapping the task should be mapped to. `Asterism` then executes each `Pegasus` task in the specified execution sites (in Figure 2, *Site A* and *Site B*), with the corresponding `dispel4py` mapping (in Figure 2, MPI and Storm), and manages data movement between sites. Note that the hybrid workflow may also contain regular `Pegasus` tasks.

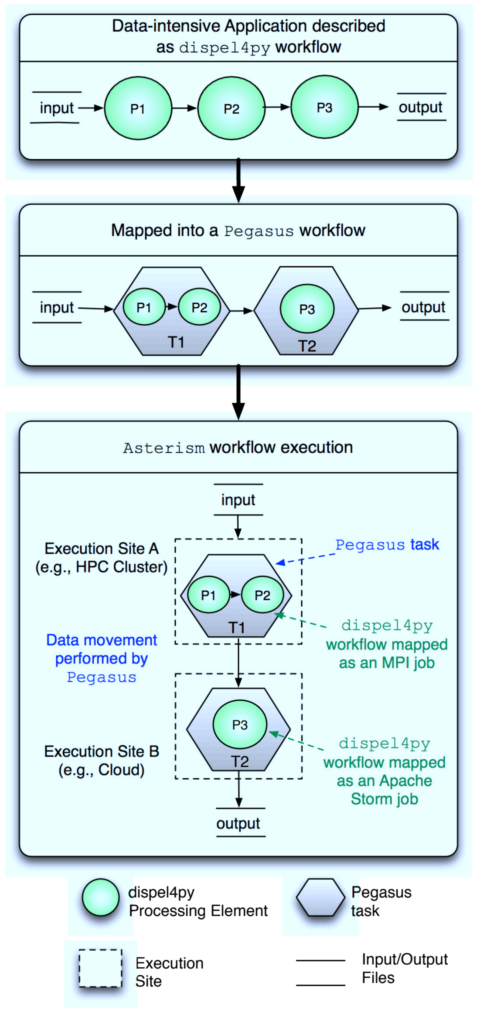

Data-intensive Application described as dispel4py workflow
input
P1
P2
P3
output
Mapped into a Pegasus workflow
input
P1
P2
T1
P3
T2
output
Asterism workflow execution
input
Execution Site A (e.g., HPC Cluster)
Pegasus task
P1
P2
T1
dispel4py workflow mapped as an MPI job
Data movement performed by Pegasus
P3
T2
Execution Site B (e.g., Cloud)
dispel4py workflow mapped as an Apache Storm job
output
dispel4py Processing Element
Pegasus task
Execution Site
Input/Output Files

Figure 2: Overview of the `Asterism` hybrid workflow framework.

4. EXPERIMENTAL EVALUATION

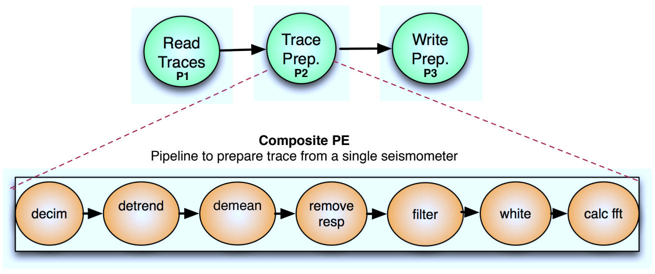
In this section, we present a practical evaluation through the instantiation of an `Asterism` workflow. We first describe the *Seismic Ambient Noise Cross-Correlation workflow* as a use case to demonstrate the feasibility of our solution—it represents a data-intensive problem commonly faced by seismology researchers; and then we describe the execution environments implemented as Docker containers.

4.1 Case Study: Seismic ambient noise cross-correlation

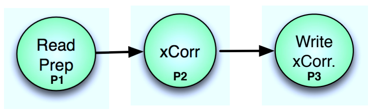
The *Seismic Ambient Noise Cross-Correlation* application represents a common data-intensive analysis pattern used by many seismologists [15]. The application preprocesses and cross-correlates traces (sequences of measurements of acceleration in three dimensions) from multiple seismic stations, and it is composed of two main phases:

- *Phase 1* (Preprocess): Each time series from a seismic station (each *trace*), is subject to a series of data-preparation treatments chosen and parameterized by seismologists; and the processing of each trace is independent from other traces. (complexity $O(n)$, where n is the number of stations).

- *Phase 2* (Cross-Correlation): For all pairs of stations compute the correlation, essentially identifying the time for signals to travel between them, and hence infer some, as it turns out time varying, properties of the intervening rock. The complexity of this phase is $O(n^2)$.



(a) Phase 1



(b) Phase 2

Figure 3: *Seismic Ambient Noise Cross-Correlation* dispel4py workflows.

Both phases, *Phase 1* and *Phase 2*, do not have to be necessarily performed in the same computing resource. In fact, scientific communities like Seismology, tend to use a wide range of e-Infrastructures for running their data-intensive applications, e.g., national cyberinfrastructures, and cloud resources, because data can be streamed to and from several e-Infrastructures to perform various analyses. Therefore, there is a need to facilitate and automate the process to run data-intensive applications across heterogeneous systems, without users making changes to their codes. In the case of the application presented here, quite often raw data is aggregated/collected (data assimilation) in a resource machine (desktop, small-middle cluster) during the period of time to perform *Phase 1* analysis, followed (in some cases) by removing the raw data if no longer required. Later, preprocessed data could be transferred to a larger resource (e.g., HPC cluster or cloud) with higher number of CPU cores and memory capacity, where *Phase 2* is applied. Additionally, it would be possible to distribute the preprocessed data to different computing resources to perform several data analyses in parallel.

The *Seismic Ambient Noise Cross-Correlation* application was originally programmed as part of the VERCE project [16] using dispel4py as shown in Figure 3—both phases run on the same computing resource. With Asterism, we can distribute the execution among heterogeneous systems, leveraging their capabilities to efficiently run the applications.

4.2 Asterism Workflow Implementation

The Asterism representation of the *Seismic Ambient Noise Cross-Correlation* workflow is composed of a Pegasus workflow (DAX) with two tasks, wrapping the *Phase1* (Figure 3a) and *Phase2* (Figure 3b) dispel4py workflows into the first and second Pegasus tasks, respectively. The DAX also contains the description of the input and output files (stations list, preprocessed data, and cross-correlation results) at a logical level, the data dependency between tasks (the preprocess task is followed by the execution of the process task), the computing sites for executing each task (MPI and Apache Storm clusters), and the mapping to run each dispel4py workflow (MPI and Apache Storm mappings, respectively).

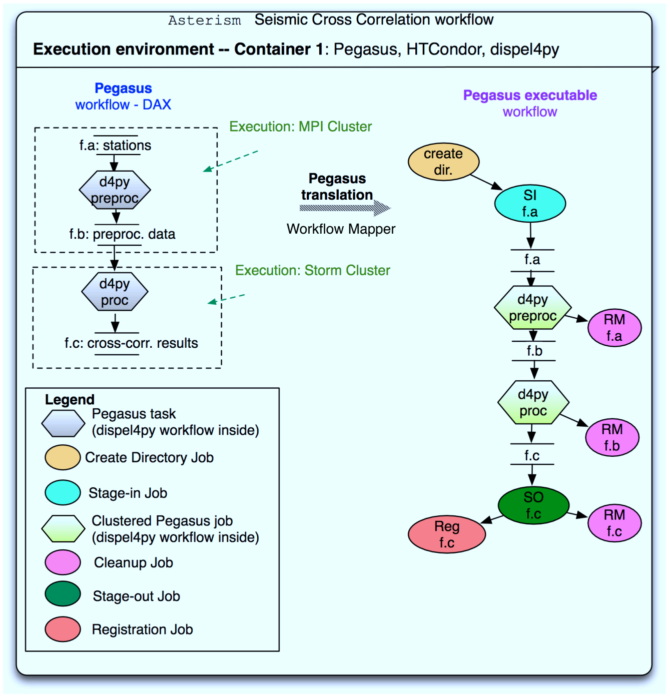


Figure 4: *Container 1* provides the Asterism Seismic Cross Correlation workflow and the execution environment necessary to run it.

During execution, the Pegasus Mapper generates an executable workflow able to run on the requested computing resources. The Mapper also creates additional jobs to stage in/out the input/output data; data cleanup nodes to remove data that is no longer required; and registration nodes to catalog output data locations for future discovery (provenance). The workflow is then submitted to DAGMan, which orchestrates the workflow execution, spawning jobs to HTCondor when all dependencies have been satisfied.

The Asterism workflow is packaged with all of its dependencies (Pegasus, HTCondor, dispel4py, etc.) into a conceptual Docker container (*Container 1*, Figure 4). To demonstrate the versatility of Asterism to run workflows in heterogeneous systems, we created two additional conceptual Docker containers: an MPI cluster (*Container 2*, Figure 5), and an Apache Storm cluster (*Container 3*, Figure 6) execution environments. *Phase1* runs in a multi-container Docker application based on *Container 2*, while *Phase2* runs in a multi-container Docker application based on *Container 3*. During execution, each dispel4py workflow is mapped to the particular execution engine indicated in the DAX (MPI or Apache Storm for this example), and data transfers between execution environments (or containers) are automatically handled by Pegasus, as well as the monitoring of the Asterism workflow execution. Data access is performed through a local shared folder with the execution system (from where Pegasus stages the data). Future work includes the full integration and management of Docker containers into Pegasus. For this experiment, the containers were manually deployed and configured. In the future, we plan to use experiment management tools such as Precip [4] to automate this step.

Container 2 (Figure 5) is configured to deploy an MPI

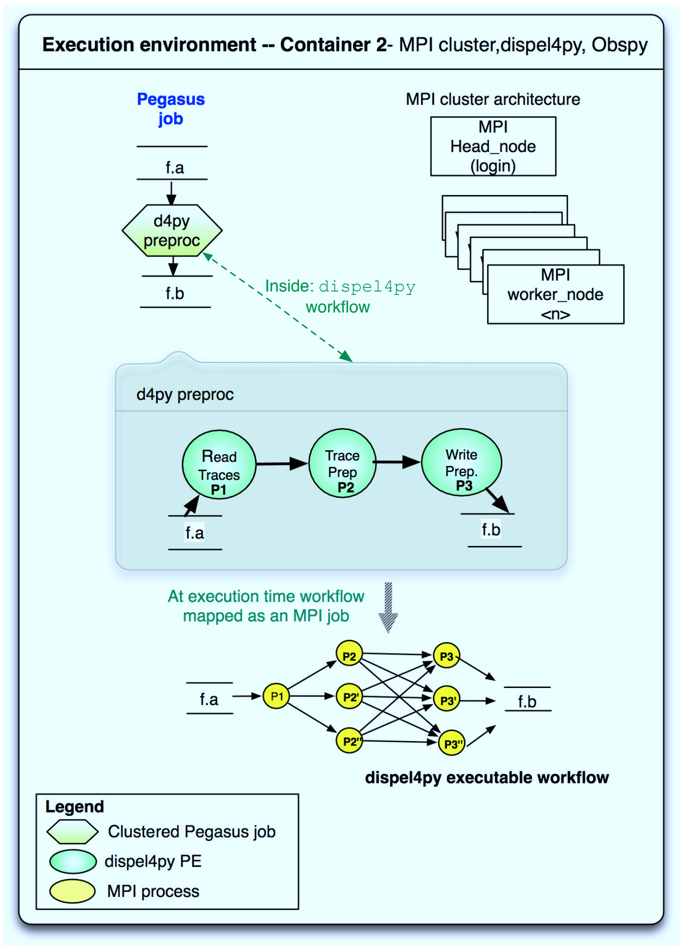


Figure 5: *Container 2* provides the execution environment necessary for running `dispel4py` workflows and MPI applications.

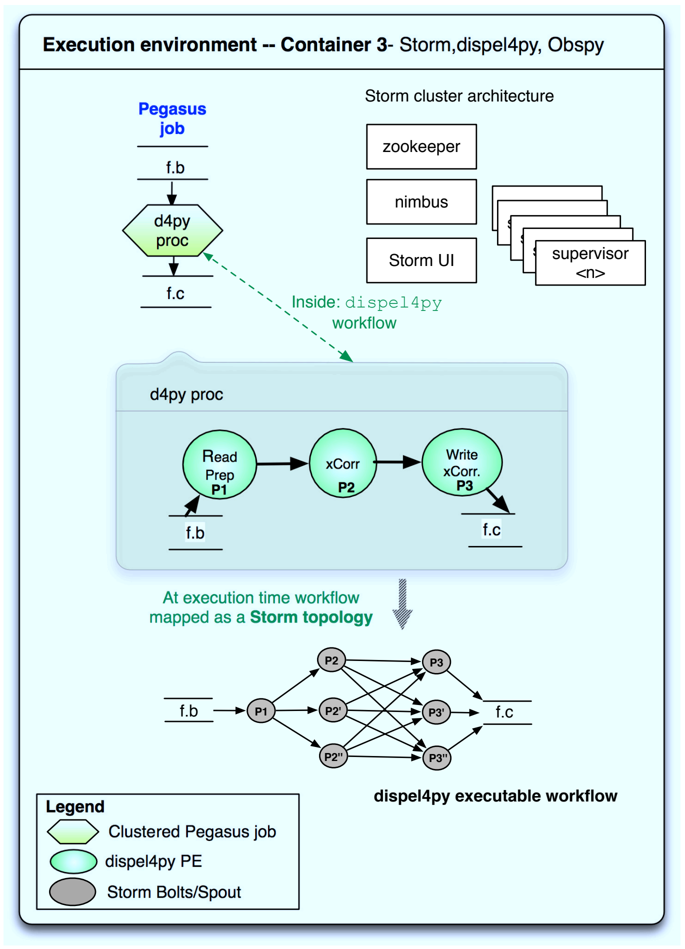

Execution environment -- Container 3- Storm,dispel4py, Obspy
Pegasus job
f.b
d4py proc
f.c
Storm cluster architecture
zookeeper
nimbus
Storm UI
supervisor <n>
Inside: dispel4py workflow
d4py proc
Read Prep P1
xCorr P2
Write xCorr. P3
f.b
f.c
At execution time workflow mapped as a Storm topology
P1
P2
P2'
P2''
P3
P3'
P3''
dispel4py executable workflow
Legend
Clustered Pegasus job
dispel4py PE
Storm Bolts/Spout

Figure 6: *Container 3* provides the execution environment necessary for running `dispel4py` workflows and Storm topologies.

cluster as a multi-container application, where one instance represents the *head node*, and the remaining instances the *worker nodes*. Additionally to the MPI capabilities, the container also provides `dispel4py` for running the *stream-based* workflows, and `ObsPy` [5], an open-source framework for processing seismological data. *Container 3* (Figure 6) also provides `dispel4py` and `ObsPy` for running the specific application phase. However, it is configured to deploy an Apache Storm cluster as a multi-container application. In this case, it provides an instance of the Apache ZooKeeper [20] (an open-source distributed configuration and synchronization service, and naming registry for large distributed systems), one instance for the master node (nimbus), one instance of the Storm user interface (Storm UI), and several supervisors (workers).

Note that these containers are interchangeable, i.e. they can run both phases of the *Seismic Ambient Noise Cross-Correlation* application. Although the explicit separation of the phases into two different execution environment is primarily to demonstrate the feasibility and flexibility of the `Asterism` approach, the execution of *Phase 2* in an Apache Storm cluster leverages the capability of the system to handle dynamic streams. `Asterism` benefits from the `docker-compose` command to automatically build dynamic virtual private networks between the nodes composing the execution cluster (MPI or Storm). Furthermore, the this command allows to scale the number of instances at runtime. A traditional MPI-like environment would not benefit of such functionality due to its static behavior. On the other hand, real-time platforms such as Apache Storm would greatly benefit of it to scale the number of computing nodes (Storm supervisors) at runtime using a single and simple command.

4.3 Asterism Workflow Execution

For running the described `Asterism` workflow we have used the academic NSF-Chamelon cloud testbed [23]. We used `CentOS7` images with Docker, and since our allocation in NSF-Chamelon was established to 40 nodes, we decided to distribute them equitably as follows: an instance of *Container 1* as the Pegasus submit node; an instance of *Container 2* as the MPI head node, 16 instances of *Container 2* as the MPI worker nodes; instances of *Container 3* as the Storm Apache ZooKeeper, nimbus, and Storm UI; and 16 instances of *Container 3* as the Storm supervisors.

The containers images were configured with Dockerfiles, which are scripts to build and configure containers, step-by-step, layer-by-layer, automatically from a source (base) image. All Dockerfiles used to generate the three types of containers are freely available online as part of the research object (RO) of this work [3]. The repositories also provide additional information such as detailed instructions on how to deploy and run the images, and the workflow. Furthermore, we linked our GitHub repositories to the *The Docker Hub* for providing Continuous Integration (CI), and allowing to directly store and share the images, without the need of build them manually in a local environment. Our repository also includes an easy-to-use script listing the commands needed to test and run the *Seismic Cross Correlation* `Asterism` workflow, and to deploy the three Docker containers described above (all the scripts and images are also available as part of the RO [3]).

The execution of the `Asterism` workflow requested an hour data from IRIS services (USArray TA [30]). The USArray TA has a list of 836 stations, though only 394 stations have online data available. The experimental evaluations (10 runs of the workflow were performed for statistical significance purpose) have reported that the time consumed by the *Phase 1* using *Container 2* is about 8 minutes, around 2 hours by the *Phase 2* using *Container 3*, and under a minute for moving data between containers. In order to evaluate long running workflows, we also ran the `Asterism` workflow continuously during 3 days, where requests for the seismic data from IRIS services were performed every two hours. Note that the scope of this work is to show the ability of `Asterism` to efficiently run a data-intensive application in heterogeneous systems with different enactment engines. Thus, we have shown that the `Asterism` framework is able to manage the entire workflow, to monitor its execution, to handle data transfers between different platforms, and to map to different enactment engines at runtime with no additional effort required from the users to run the application in different contexts. `Asterism`, therefore, provides an absolutely abstraction of the selected compute resource(s) and their technical details. For maximizing the application's performance, other configurations could be considered (e.g., running both phases in the MPI cluster, or increasing the number of Storm Supervisors).

5. DATA-INTENSIVE WORKFLOWS AS A SERVICE

The `Asterism` framework combined with Docker containers provide an integrated, complete, easy-to-use, portable approach to run data-intensive workflow applications on distributed platforms. The three containers designed and implemented for the use case described in this paper (Figures 4, 5, and 6) integrate the "Data-Intensive workflows as a service" (`DIaaS`) model that delivers data-intensive workflow applications over the wide-area. In a `DIaaS` model, the framework delivers specialized software to execute data-intensive applications in a scalable, efficient, and robust manner. Since all the software needed for running `Asterism` workflows and their dependencies are packed in `DIaaS` model, it substantially reduces the time (and the possible human errors) spend by scientists to build such systems by themselves, which consequently allow them to focus in their research. Note that many *research campaigns* develop and refine models of the phenomena of interest by using numerical simulations to expose the implications of mathematical models intricate combinations with observations that provide initial conditions. Analysis of the differences between synthetic and observed values leads to corrections that need to be propagated into the model. Such methodological patterns are pursued repeatedly for sets of initial conditions and multiple observation comparisons until the model reaches stability or the limit of resolution in the context being studied. `Asterism DIaaS` model improves the productivity for such campaigns by being able to encode and enact more aspects of a research campaign into one workflow by using an easy-to-use data-intensive model.

The most common use of the `Asterism DIaaS` framework will be the execution of workflow applications on virtualized environments such as cloud computing, due to its ability to easily scale the number of computing resources (e.g., MPI workers or Storm supervisors). Users can then download and deploy the `Asterism` framework in any cloud platform, or computing environment (where Docker is supported), and extend our base images for running their own data-intensive applications seamlessly.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented `Asterism`, an integrated and complete approach for running data-intensive workflow applications on distributed heterogeneous systems. `Asterism` provides a framework into which an ever-growing range of target platforms will fit. It carefully avoids the encoding of the research methods being shaped by these targets, as they are evolving rapidly.

By leveraging the capabilities of two widely used workflow management systems (`Pegasus` and `dispel4py`), we developed a hybrid workflow approach to enable the execution of data-intensive *stream-based* workflow applications across different e-Infrastructures. The feasibility of the approach was evaluated using a *seismic ambient noise cross-correlation* application, which is a real data-intensive workflow, and heterogeneous e-Infrastructures described as Docker containers, which were deployed and executed in an academic cloud. From the containers, we derived a Data-Intensive workflow as a Service (`DIaaS`) model to enable easy composition and deployment of data-intensive workflows on cloud platforms.

Future works include the development of a portfolio of e-Infrastructures to be distributed as part of the `Asterism DIaaS` framework besides Apache Storm and the MPI-based cluster. To ease the framework usage, we also plan to provide `Asterism` via an experiment management tool (e.g., Precip [4]), where the entire infrastructure would be deployed automatically using a single and simple command—the user will be prompted with basic questions (e.g., the number of worker nodes, etc.), and the tool would deliver the requested environment. Finally, we also intend to distribute `Asterism` as part of our community resources for enabling research on scientific workflows [14].

Acknowledgments

This work was carried out when Rosa Filgueira worked for the University of Edinburgh. It was funded under the Scottish Informatics and Computer Science Alliance with the Postdoctoral and Early Career Researcher Exchanges fellowship, and by the National Science Foundation under the SI^2–SSI program, award number 1148515. We thank the NSF Chameleon Cloud for providing time grants to access their resources.

7. REFERENCES

[1] M. Albrecht, P. Donnelly, et al. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, page 1. ACM, 2012.

[2] M. D. Assuncao, R. N. Calheiros, et al. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79-80:3–15, 2015.

[3] Asterism: Research Object. https://scitech.isi.edu/ro/asterism.

[4] S. Azarnoosh, M. Rynge, et al. Introducing precip: an api for managing repeatable experiments in the cloud. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 2 of *CloudCom*, pages 19–26, 2013.

[5] M. Beyreuther, R. Barsch, et al. Obspy: A python toolbox for seismology. *Seismological Research Letters*, 81(3):530–533, 2010.

[6] C. Boettiger. An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79, Jan. 2015.

[7] R. Chard, K. Chard, et al. Cost-aware cloud provisioning. In *2015 IEEE 11th International Conference on e-Science (e-Science)*, pages 136–144. IEEE, 2015.

[8] E. Deelman, K. Vahi, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46(0):17–35, 2015.

[9] E. Deelman, K. Vahi, et al. Pegasus in the cloud: Science automation through workflow technologies. *IEEE Internet Computing*, 20(1):70–76, 2016.

[10] Docker. https://www.docker.com.

[11] Docker hub. https://hub.docker.com/.

[12] R. Evans. Apache storm, a hands on tutorial. In *2015 IEEE International Conference on Cloud Engineering (IC2E)*, pages 2–2, 2015.

[13] T. Fahringer, R. Prodan, et al. Askalon: A development and grid computing environment for scientific workflows. In *Workflows for e-Science*, pages 450–471. 2007.

[14] R. Ferreira da Silva, W. Chen, et al. Community resources for enabling and evaluating research on scientific workflows. In *2014 IEEE 10th International Conference on e-Science*, eScience'14, pages 177–184, 2014.

[15] R. Ferreira da Silva, E. Deelman, et al. Automating environmental computing applications with scientific workflows. In *Environmental Computing Workshop (ECW'16)*, 2016.

[16] R. Filgueira, A. Krause, et al. dispel4py: An agile framework for data-intensive escience. In *Proc. 11th IEEE eScience Conf.*, 2015.

[17] R. Filgueira, A. Krause, et al. dispel4py: A python framework for data-intensive scientific computing. *International Journal of High Performance Computing Applications (IJHPCA)*, pages 9–16, 2016.

[18] J. Frey. Condor dagman: Handling inter-job dependencies. *University of Wisconsin, Dept. of Computer Science, Tech. Rep*, 2002.

[19] W. Gerlach, W. Tang, and A. Wilke. Container orchestration for scientific workflows. In *2015 IEEE International Conference on Cloud Engineering*, 06/2015 2015.

[20] P. Hunt, M. Konar, et al. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference*, volume 8, page 9, 2010.

[21] A. Jain, S. P. Ong, et al. Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059, 2015.

[22] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. ÃĄlvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. Badia. Servicess: An interoperable programming framework for the cloud. *Journal of Grid Computing*, 12(1):67–91, 2014.

[23] J. Mambretti, J. Chen, et al. Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn). In *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, pages 73–79, 2015.

[24] M. Nardelli. A framework for data stream applications in a distributed cloud. In *Proceedings of the 8th ZEUS Workshop, Vienna, Austria, January 27-28, 2016.*, pages 56–63, 2016.

[25] Nextflow. http://www.nextflow.io/index.html.

[26] Open mpi: Open source high performance computing. https://www.open-mpi.org.

[27] Pegasus and LIGO. http://pegasus.isi.edu/2016/02/11/pegasus-powers-ligo-gravitational-waves-detection-analysis.

[28] I. J. Taylor, E. Deelman, et al. *Workflows for e-Science: scientific workflows for grids.* 2014.

[29] D. Thain, T. Tannenbaum, et al. Distributed computing in practice: the condor experience. *Concurrency and computation: practice and experience*, 17(2-4):323–356, 2005.

[30] USArray TA. http://ds.iris.edu/ds/nodes/dmc/earthscope/usarray/_US-TA-operational.

[31] VERCE. http://www.verce.eu.

[32] K. Vukojevic-Haupt, F. Haupt, et al. Bootstrapping complex workflow middleware systems into the cloud. In *2015 IEEE 11th International Conference on e-Science (e-Science)*, pages 126–135. IEEE, 2015.

[33] C. Walter. Kryder's Law: The doubling of processor speed every 18 months is a snail's pace compared with rising hard-disk capacity, and Mark Kryder plans to squeeze in even more bits. *Scientific American*, pages 32–33, August 2005.

[34] K. Wolstencroft, R. Haines, et al. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic acids research*, 41(Web Server issue):W557–W561, 2013.

[35] C. Zheng and D. Thain. Integrating containers into workflows: A case study using makeflow, work queue, and docker. In *8th International Workshop on Virtualization Technologies in Distributed Computing*, pages 31–38. ACM, 2015.