# Parallelization of Scientific Workflows in the Cloud

Ji Liu, Esther Pacitti, Patrick Valduriez, Marta Mattoso


# Parallelization of Scientific Workflows in the Cloud

**Ji Liu, Esther Pacitti, Patrick Valduriez, Marta Mattoso**

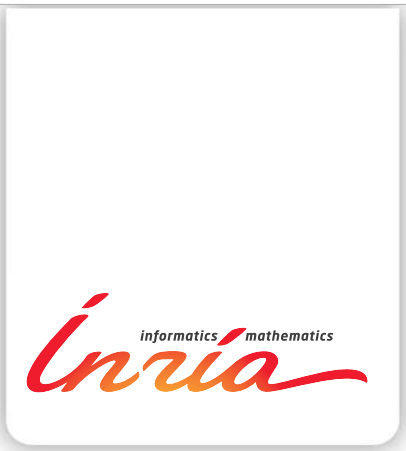
**RESEARCH REPORT**

**N° 8565**

March 2014

Project-Teams ZENITH

ISSN 0249-6399 ISRN INRIA/RR--8565--FR+ENG




# Parallelization of Scientific Workflows in the Cloud*

Ji Liu[†‡§], Esther Pacitti[§‡], Patrick Valduriez[¶‡], Marta Mattoso[‖]

Project-Teams ZENITH

Research Report n 8565 — March 2014 — 50 pages

**Abstract:** Nowadays, more and more scientific experiments need to handle massive amounts of data. Their data processing consists of multiple computational steps and dependencies within them. A *data-intensive scientific workflow* is an appropriate tool for modeling such process. Since the execution of data-intensive scientific workflows requires large-scale computing and storage resources, a cloud environment, which provides virtually infinite resources is appealing. However, because of the general geographical distribution of scientific groups collaborating in the experiments, multisite management of data-intensive scientific workflows in the cloud is becoming an important problem. This paper presents a general study of the current state of the art of data-intensive scientific workflow execution in the *cloud* and corresponding multisite management techniques.

**Key-words:** scientific workflow, scientific workflow management system, cloud, multisite cloud, distributed and parallel data management, scheduling.

* Work partially funded by CNPq, CAPES, FAPERJ, INRIA (Hoscar and Music projects) and Microsoft (Zcloudflow project), and performed within the Institut de Biologie Computationnelle (www.ibc-montpellier.fr).

† MSR-INRIA Joint Centre
‡ LIRMM
§ University Montpellier 2
¶ INRIA
‖ COPPE/Federal University of Rio de Janeiro

**RESEARCH CENTRE**
**SOPHIA ANTIPOLIS – MÉDITERRANÉE**

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

# Parallélisation de Workflows Scientifiques dans le Cloud rapport de recherche Inria

**Résumé :** De nos jours, les expériences scientifiques doivent manipuler très souvent de très grands masses de données. Le traitement de ces données comporte plusieurs étapes de calcul inter-dépendantes. Un *workflow scientifique orienté-données* est un outil pratique pour la modélisation de ce processus. L'exécution de ces workflows nécessite des ressources de calcul et de stockage à grande échelle, et le *cloud*, qui fournit des ressources presque infinies dans un data center, devient intéressant. Toutefois, en raison de la distribution géographique des différents groupes scientifiques qui peuvent collaborer dans les expérimentations, la gestion de l'aspect multisite dans le cloud devient un problème important. Ce rapport présente une étude générale de l'état de l'art pour l'exécution des workflows scientifique orientés-données dans le cloud et les techniques correspondantes pour la gestion de multisite.

**Mots-clés :** workflow scientifique, système de gestion de workflow scientifique, cloud, cloud multisite, gestion de données distribuées et paralèles, ordonnancement.

# Summary

# 1 Introduction

Many large-scale scientific experiments take advantage of scientific workflows to model data operations such as loading input data, data processing, data analysis, and aggregating output data. Scientific workflows allow scientists to easily model and express the entire data processing steps and their dependencies, typically as a directed graph or Directed Acyclic Graph (DAG). As more and more data is consumed and produced in modern scientific experiments, scientific workflows become data-intensive. In order to process large-scale data within reasonable time, they need to be executed with parallel processing techniques in the grid or the cloud.

A *Scientific Workflow Management System* (*SWfMS*) is an efficient tool to execute workflows and manage data sets in various computing environments. A SWfMS can exploit diverse parallel technologies to harness a lot of computing and storage resources. It can also produce *provenance data* [42] during the workflow execution. Several SWfMSs such as Pegasus [34], Swift [123], Kepler [6], Taverna [86], Galaxy [47] and Chiron [84] are now used intensively by various research communities, e.g. astronomy, biology, computational engineering.

The ability to provide elastic computing and storage resources makes cloud computing a good option for scientific workflow management, including development and execution. The cloud makes use of virtualization techniques to offer diverse scalable services no matter what physical machines are used. The diverse services include three main kinds: *Infrastructure as a Service* (*IaaS*), *Platform as a Service* (*PaaS*) and *Software as a Service* (*SaaS*). Since the quality of these services is guaranteed by *Service Level Agreement* (*SLA*), the cloud offers a relatively stable execution environment for scientific workflows. In a pay-as-you-go manner, cloud computing also provides an appealing alternative to owning specific physical resources for executing data-intensive scientific workflows. Furthermore, cloud storage makes cloud computing appropriate for data-intensive workflow execution.

In general, one cloud site is sufficient for executing one user application. However, in the case of scientific workflows, some important restrictions may force their execution in a *multisite cloud*, i.e. a cloud with multiple distributed data centers, each being explicitly accessible to cloud users. For instance, some activities need to be executed at a cloud site trusted by the scientists for workflow monitoring without malicious attack; some activities need to be moved to another cloud site because of stored big input data at that site and the cost of transferring this big data to another site is very high; some activities have to be executed at a site where more computing resources are available; some other activities may invoke special programs (instruction data), which are located at a specific cloud site and cannot be moved to another site because of proprietary reasons, i.e. proprietary restriction. The configuration data, which includes workflow representation files or workflow parameters, can be located at one site or distributed at different sites. In this situation, multisite cloud is appealing for data-intensive scientific workflows.

There have been a few surveys of techniques for SWfMSs. Bux and Leser [12] provide an overview of parallelization techniques for SWfMSs, including their implementation in real systems. They also discuss major improvements to the landscape of SWfMS. Yu and Buyya [119] examine the existing SWfMSs designed for grid computing. They propose taxonomies for different aspects of SWfMSs, including workflow design, information retrieval, workflow scheduling, fault tolerance and data movement.

In this paper, we focus on data-intensive scientific workflows and the multisite cloud environment. The main contributions of this paper are:

1. We propose a five-layer SWfMS functional architecture, which is useful to discuss the techniques for data-intensive scientific workflows. This architecture can also be a baseline for other work and help on the assessment and comparison of SWfMSs.

2. We introduce workflow parallelization techniques and scientific workflow scheduling algorithms, and give a comparative analysis of the existing solutions.

3. We introduce the basic techniques for the parallel execution of scientific workflows in the cloud, including multisite management and data storage. And we identify research issues for improving the execution of data-intensive scientific workflows in a multisite cloud

This document is organized as follows. Section 2 gives an overview of scientific workflow management, including system architectures and basic functionality. Section 3 focuses on the techniques used for parallel execution of scientific workflows. Section 4 introduces cloud computing, and discusses the basic techniques for the parallel execution of scientific workflows in multisite cloud. Section 5 summarizes the main findings of this study and discusses the open issues raised for executing data-intensive scientific workflows in multisite cloud.

# 2 Scientific Workflow Management

This section introduces scientific workflow management, including scientific workflows and systems. First, we present related notions of scientific workflows and SWfMSs. Then, we detail the functional architecture and the corresponding functionality of SWfMSs. Finally, we discuss the features and techniques for data-intensive workflows used in SWfMSs.

## 2.1 Introduction

A SWfMS manages a scientific workflow all along its life cycle. This section introduces the related concepts of SWfMSs by four parts: scientific workflow, scientific workflow life cycle, SWfMS and workflow examples.

### 2.1.1 Scientific Workflows

A workflow is the automation of a process, during which data is processed by different logical data processing activities according to a set of rules. Workflows can be divided into business workflows and scientific workflows. Business workflows are widely used for business data processing. According to the workflow management coalition, a business workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [20]. A business process is a set of one or more linked procedures or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships [20]. Business workflows make business processes more efficient and more reliable.

Different from business workflows, scientific workflows are typically used for modeling and running scientific experiments. Scientific workflows can assemble scientific data processing activities and automate the execution of these activities to reduce the makespan, which represents the entire workflow execution time. A scientific workflow is the assembly of complex sets of scientific data processing activities with data dependencies between them [31]. Scientific workflows can be represented in different ways. The most general representation is a directed graph, in which nodes correspond to data processing activities and edges represent the data dependencies. But most often, a scientific workflow is represented as a DAG or even as a sequence (pipeline) of activities which is sufficient for many applications. Directed Cyclic Graphs (DCG) are harder to support since iteration is needed to represent repeated activities, e.g. with a whiledo construct [119].

Although business workflows and scientific workflows have some similarities, they are quite different. The first difference is the interaction with participants. In business workflows, data can be processed by different participants, which can be data processing machines or humans. In scientific workflows, data is processed only by machines while the scientists just need to monitor the workflow execution or control execution when necessary. The interaction of humans during the execution of scientific workflows is much less than that of business workflows. The second difference lies in the data flows and control flows [118]. Business workflows focus on procedural rules that generally represent the control flows while scientific workflows highlight data flows that are depicted by data dependencies. This is reasonable since scientific experiments may need to deal with big experimental data. A data-intensive scientific workflow is a scientific workflow that processes, manages or produces huge amounts of data during execution.

### 2.1.2 Scientific Workflow Life Cycle

The life cycle of a scientific workflow is a description of the state transitions of a scientific workflow from creation to completion [31, 50]. A scientific workflow life cycle generally contains four phases. Görlach *et al.* [50] propose that a scientific workflow life cycle contains modeling phase, deployment phase, execution and monitoring phase, and analysis phase. Deelman *et al.* [31] argue that a scientific workflow life cycle consists of composition phase, mapping phase, execution phase and provenance phase. In general, a scientific workflow life cycle can be divided into four phases: composition phase, deployment phase, execution phase and analysis phase:

1. The composition phase is for the creation of an abstract scientific workflow. An abstract scientific workflow is defined by the functionality of each activity and data dependencies between activities. Workflow composition can be done through a textual or Graphical User Interface (GUI). SWfMS users can reuse the existing scientific workflows with or without modification [55].

2. The deployment phase is for constructing a concrete scientific workflow, which consists of concrete methods (and associated codes) for each functional activity defined in the workflow composition phase, so that the workflow can be executed.

3. The execution phase is for the execution of scientific workflows with associated data, during which input data is processed and output data is produced. Generally, the provenance data (see next section) is produced in this phase.

4. The analysis phase is to apply the output data to scientific experiments, to analyze workflow provenance data and to share the workflow information.

Using provenance data makes it possible to integrate all phases of the scientific workflow life cycle [76], thus making provenance an important functionality of SWfMS.

An activity is a description of a piece of work that forms one logical step within a scientific workflow representation. In a scientific workflow, an activity defines the associated data formats and data processing methods but it requires associated data and computing resources to carry out execution. The associated data in an activity consists of input data and configurable parameters. When the configurable parameters are fixed and the input data is provided, the execution of a workflow activity is represented by several tasks. A task is the representation of an activity within a one-time execution of this activity, which processes a data chunk. An activity can correspond to a set of tasks for different parts of input data. Sometimes, the aforementioned tasks are called "jobs" [15] while the term "task" is used for representing the concept of a data processing activity that composes a scientific workflow [12].

### 2.1.3 Scientific Workflow Management Systems

A Workflow Management System (WfMS) is a system that defines, creates, and manages the execution of workflows. A WfMS is able to interpret the workflow process definition typically in the context of business applications. A SWfMS is a WfMS that handles and manages scientific workflow execution. It is powerful tool to execute workflows in a scientific workflow engine, a software service that provides the runtime environment for workflow execution. In order to execute a scientific workflow in a given environment, a SWfMS typically generates a Workflow Execution Plan (WEP), which is a program that captures optimization decisions and execution directives, typically the result of compiling and optimizing a workflow, before execution.

To handle scientific workflows, SWfMS should support additional functionality such as workflow provenance. Workflow provenance may be as (or more) important as the scientific experiment itself [42]. Provenance is the metadata that captures the derivation history of a dataset, including the original data sources, intermediate datasets, and the workflow computational steps that were applied to produce this dataset [23, 26, 49, 56]. Furthermore, provenance data can be used for workflow analysis and workflow reproducibility.

### 2.1.4 Scientific Workflow Examples

Scientific workflows have been used in various scientific domains. In the astronomy domain, Montage[1] is a computing and data-intensive application that can be modeled as a scientific workflow initially defined for the Pegasus SWfMS. This application is the result of a national virtual observatory project that stitches tiles of images of the sky from diverse sky surveys into a photorealistic single image [33]. Montage is able to handle a wide range of astronomical image data including the Two Micron All Sky Survey, (2MASS[2]), the Digitized Palomar Observatory Sky Survey, (DPOSS[3]), and the Sloan Digital Sky Survey (SDSS[4]) [60]. Each survey possesses huge amounts of data and covers a corresponding part of sky in visible wavelengths or near-infrared wavelengths. 2MASS has roughly 10 terabytes, DPOSS has roughly 3 terabytes and SDSS contains roughly 7.4 terabytes. All the data can be downloaded from a corresponding server at the aforementioned links and then staged into the execution environment, such as a shared-disk file system or a database, in order to be processed.

The structure of a small montage workflow is shown in Figure 1. The number within a node represents the level of the activity in the workflow. Level one is for orphan activities (with no parent activities). The level of any other activity is the result of the maximum level of any of its parents plus one. At the same level, all activities are independent of each other [101] while each level consists of the same program working on different input data. The activities in the first level exploit an mProject program that projects a single image to the scale defined in a pseudo-FITS header template file (an ASCII file with the output image header lines, but not padded to 80 characters and with newlines at the end of each line). The second level activities utilize an mDiffFit program to create a table of image-to-image difference parameters. The third level activity takes advantage of an mFitplane program to fit the images generated by former activities to an image. The fourth level activity uses an mBgModel program to interactively determine a set of corrections to apply to each image to achieve a "best" global fit according to the image-to-image difference parameter table. The fifth level activities remove a background from a single image through an mBackground program. The sixth level activity employs an mImgtbl program to extract the FITS header information (information about one or more scientific coordinate

[1]Montage project: http://montage.ipac.caltech.edu/
[2]2MASS: http://www.ipac.caltech.edu/2mass/
[3]DPOSS: http://www.astro.caltech.edu/~george/dposs/
[4]SDSS: http://www.sdss.org/

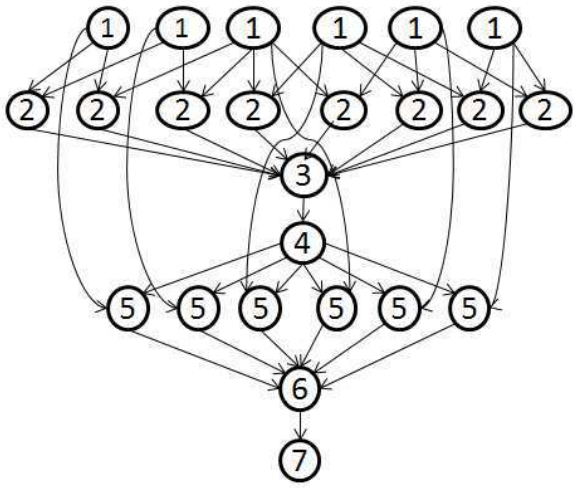



Figure 1: **The structure of a small Montage workflow [101].**

systems that are overlaid on the image itself) from a set of files and to create an ASCII image metadata table. Finally the last activity pieces together the projected images using the uniform FITS header template and the information from the same metadata table generated by the previous activity. This activity applies an mAdd program.

In the bioinformatics domain, SciEvol [82] is a workflow for molecular evolution reconstruction that aims at inferring evolutionary relationships on genomic data. To be executed in the Chiron SWfMS, SciEvol consists of 12 activities as shown in Figure 2. The first activity (pre-processing a FASTA file) is a Python script to format the multi-fasta input file by deleting the stop codons in sequences. FASTA file is a textual presenting format for nucleotide sequences or peptide sequence. The second activity (MSA construction) constructs a Multiple Sequence Alignment (MSA) using a MAFFT program (or other MSA programs). A MAFFT program is generally for generating the alignment of three or more biological sequences (protein or nucleic acids) of similar length. This activity receives a pre-formatted multi-fasta file as input and produces the MSA in FASTA format as output. The third activity (MSA conversion) executes ReadSeq to convert the MSA in FASTA format to that in another format called PHYLIP, which is used in the phylogenetic tree construction activity. The fourth activity (pre-processing PHYLIP file) formats the input PHYLIP format file (referenced as "phylip-file-one") according to the format definition and generates a second PHYLIP format file (referenced as "phylip-file-two"). The fifth activity (tree construction) receives the "phylip-file-one" as input and produces a phylogenetic tree in Newick format [40] as output. The sixth activities (evolutionary analysis from 6.1 – 6.6) analyze the phylogenetic tree by corresponding parameters and they generate a set of files containing evolutionary information as output. The last activity (data analysis) automatically processes the output files obtained from the previous activities.

There are many other data-intensive workflows in bioinformatics. For instance, SciPhylomics [87] is designed for producing phylogenomic trees based on an input set of protein sequences of genomes to infer evolutionary relationships among living organisms. SciPPGx [36] is a computing and data-intensive pharmacophylogenomic analysis workflow for providing thorough inferring support for pharmacophylogenomic hypotheses. SciPhy [81] is used to construct phylogenetic trees from a set of drug target enzymes found in protozoan genomes. All these bioinformatics workflows have been executed using SciCumulus SWfMS [27].

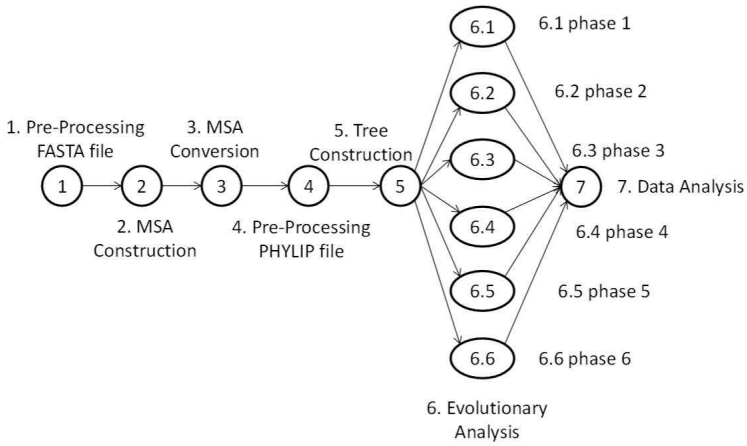



Figure 2: **SciEvol workflow [82].**

## 2.2 Functional Architecture of SWfMSs

The functional architecture of a SWfMS can be layered as follows [34, 123, 6, 84]: presentation, user services, WEP generation, WEP execution and infrastructure. Figure 3 shows this architecture. The higher layers take advantage of the lower layers to realize more concrete functionality. A user interacts with a SWfMS through presentation and realizes the desired functions at user services layer. A scientific workflow is processed at WEP generation layer to produce a WEP, which is executed at the WEP execution layer. The SWfMS accesses the physical resources through the infrastructure layer for scientific workflow execution.

### 2.2.1 Presentation Layer

The presentation layer serves as a User Interface (UI) for the interaction between users and SWfMSs at all stages of the scientific workflow life cycle. The UI can be textual or graphical. This interface is responsible for designing a workflow by assembling data processing activities linked by dependencies. This layer also supports the functionality of showing execution status, expressing workflow steering and information sharing commands.

The language for the textual interface is largely used for designing scientific workflows in SWfMSs. Different from batch scripts, the textual language supports parallel computations on distributed computing and storage resources. The configuration or administration becomes complicated in this environment while the language defined by a SWfMS should be easy to use. Most SWfMS languages support the specification of a workflow in a DAG structure while some SWfMS languages also support iteration for DCG.

Wilde *et al.* [115] propose a distributed parallel scripting language called Swift. Swift supports workflow specifications in both DAG and DCG. It is a C-like syntax that describes data, data flows and applications by focusing on concurrent execution, composition and coordination of independent computational activities. Variables are used in Swift to name the local variables, arguments, and returns of a function. The variables in Swift have three types: primitive, mapped, and collection. Primitive variables have the basic data structures such as integer, float, string, Boolean and array. Mapped variables refer to files external to the Swift script. Collection variables are in the structures that contain a set of variables, such as arrays. Swift operations have three categories: built-in functions, application interface functions and compound func-

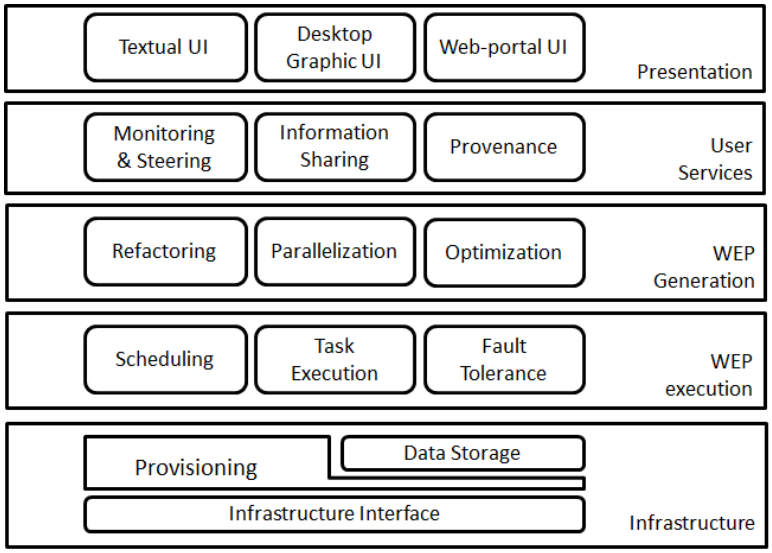



Figure 3: **Functional architecture of a SWfMS.**

tions. Built-in functions are implemented by the Swift runtime system to perform various utility functions such as numeric conversion, string manipulation, etc. An application interface function provides the information to the Swift runtime system to invoke a program. A compound function is a function that invokes other functions.

Pegasus uses Wings to create scientific workflows [46]. The workflows are created through three stages in Pegasus/Wings: the first stage specifies the abstract structure of the workflow and creates a workflow template; the second stage specifies what data to be used in the workflow and creates a workflow instance; the third stage specifies the data replicas and their locations to form an executable workflow. The later stage is done by Pegasus while the first two are realized by Wings. In Wings, workflow and its activities are represented as semantic objects. The programs are represented as workflow components to process data, which is represented as individual files or file collections. An activity is represented as a node that contains a set of computations, which may contain one computation component or a collection of computations. The data dependencies are represented as links to carry data from or to a workflow node. After the presentation of programs, activities and data dependencies, a workflow template is created. With the binding of input data sets, a workflow instance is generated as a DAG in XML format. Then, Pegasus automatically maps the workflow instance to distributed computing nodes to form an executable workflow and manages workflow execution.

Chiron [84] also represents the scientific workflow activities and dependencies as a DAG in XML textual format and execute the activities according to the activity dependencies. Ogasawara *et al.* [83] propose an algebraic language implemented in Chiron to encapsulate the workflow activities in six operators: Map, SplitMap, Reduce, Filter, SRQuery and JoinQuery. The Map operator consumes and produces a basic data chunk, which represents the data chunk that has a smallest amount of data while it contains all the necessary data to be processed in an activity. The SplitMap operator consumes a basic data chunk while it produces several basic data chunks. The Reduce operator reduces several basic data chunks to one basic data chunk. The Filter operator removes useless data chunks. SRQuery and MRQuery are traditional relational algebra expressions. Each activity corresponds to an operator. During workflow execution,

these operators are able to parallelize the workflow execution onto the distributed computation resources.

Taverna utilizes a simple conceptual unified flow language (Scufl) to represent the workflow [86]. Scufl is an XML-based language, which consists of three main entities: processors, data links, and coordination constraints. Processors represent a computational activity in a scientific workflow. Data links and coordination constraints separately represent the data dependencies and control dependencies between two activities.

SWfMSs such as Galaxy [47], Taverna [86] and Kepler [6] offer a GUI for workflow design. The GUI simplifies the designing process of a scientific workflow for the SWfMS users to assemble the workflow components described as icons through drag-and-drop functionality. Graphical SWfMSs combine the efficiency of scientific workflow design and the ease of scientific workflow representation. Desktop-based graphical SWfMSs are typically installed either in a local computer or in a remote server that is accessible through network connection. The local computer or remote server can be connected to large computing and storage resources for large-scale scientific workflow execution. Some graphical SWfMSs such as Galaxy are web-portal-based, which makes it easy to share workflow information among the SWfMS users. With these SWfMSs, a scientific workflow is generally designed in a browser on the client side but executed in a private or public web server. Some of the graphical SWfMSs take textual languages as inner representation of a scientific workflow. For instance, Taverna utilizes Scufl within the SWfMS while Galaxy represents workflows in JSON format [1].

### 2.2.2 User Services Layer

The user services layer is responsible for supporting user functionality, i.e. workflow monitoring and steering, workflow information sharing and providing workflow provenance data.

Scientific workflow monitoring makes it possible to get real-time execution status for SWfMS users. Since scientific workflow execution may take a long time, dynamic monitoring and steering of the execution are important to control workflow execution [31]. Workflow monitoring tracks the execution status and displays this information to users during workflow execution [20]. Through workflow monitoring, a scientist can verify if the result is already enough to prove her hypothesis [23]. Workflow monitoring remains an open challenge as it is hard to fully support. However, it can be achieved based on log data (in log files) or more general provenance data, typically in a database [75]. Gunter *et al.* [52] and Samak *et al.* [98] propose the Stampede monitoring infrastructure for real-time workflow monitoring and troubleshooting. This infrastructure takes a common data model to represent scientific workflow execution and utilizes a high-performance loader to normalize the log data. It offers a query interface for extracting data from the normalized data. It has been initially integrated with Pegasus SWfMS and then adapted in Triana SWfMS [110]. Horta *et al.* [56] propose a provenance interface to describe the production and consumption relationships between data artifacts such as output data files and computational activities at runtime for workflow monitoring. This interface can be used to select the desired output data to monitor the workflow execution for SWfMS users through browsers or a high-resolution tiled display. This interface is based on on-line provenance query supported by algebraic approach. The on-line provenance query is different from the provenance collected at runtime, but made available only after the execution, where monitoring is no longer possible. This interface is available for Chiron [84] or SciCumulus [27] that store all the provenance data in a relational database. SciCumulus is an extension of Chiron for cloud environments.

Workflow steering is the interaction between a SWfMS and a user to control the workflow execution progress or configuration parameters [49]. Through workflow steering, a scientist can control workflow execution dynamically so that she does not need to continue unnecessary

execution or execute a scientific workflow again when an error occurs [23, 24]. Scientific workflow steering, which still remains an open issue, saves much time for SWfMS if users are able to analyze the execution state at runtime to check if the result is already enough for the experiment or even make some changes at runtime instead of executing the workflow again [37].

Through workflow information sharing, SWfMS users can share workflow information including workflow design, the input data or the output data. Since designing a scientific workflow is challenging work and workflow reusing can reduce repetitive work between different scientist groups, it is useful to share workflow information among SWfMS users. A SWfMS can directly integrate workflow sharing functionality in the system [54] or exploit social network to support workflow information sharing [47]. Moreover, the workflow sharing across different SWfMSs can be achieved through a framework in abstract (abstract scientific workflow) and concrete level (concrete scientific workflow) [93].

Provenance data in scientific workflows is important to support reproducibility, result interpretation and problem diagnosis. Provenance data management concerns the efficiency and effectiveness of collecting, storing, representing and querying provenance data. Different methods have been proposed for different SWfMSs. Gadelha *et al.* [61] develop MTCProv, a provenance component for the Swift SWfMS. Swift optionally produces provenance information in its log files while this data is exported to relational databases by MTCProv. MTCProv supports a data model for representing provenance and provides a provenance query interface for the visualization of provenance graphs and querying of provenance information. Kim *et al.* [67] present a semantic-based approach to generate provenance information in the Wings/Pegasus framework. Wings is a middleware that supports the creation of workflow templates and instances, which are then submitted to Pegasus. This approach produces activity-level provenance through the semantic representations used in Wings, and execution provenance through Pegasus' task scheduling and execution process. SPARQL (SPARQL Protocol and RDF Query Language), a semantic query language, is used for querying provenance data. Costa *et al.* [23] propose PROV-Wf, a practical approach for capturing and querying provenance data for workflows. PROV-Wf gathers provenance data in different granularities based on PROV recommendation [9]. The PROV-Wf contains three main parts: the structure of the experiment, execution of the experiment and environment configuration. PROV-Wf supports prospective and retrospective provenance data allowing for on-line provenance queries through SQL. The provenance database of this approach acts as a statistics catalog from DBMS. Altintas *et al.* [4] present a provenance information collection framework for Kepler. This framework can collect provenance information thanks to its implementation of event listener interfaces. Moreover, Crawl *et al.* [25] introduce a provenance system that manages provenance data from Kepler. This system records both data and dependencies of tasks executing on the same computing node. The provenance data is stored in a MySQL database. The Kepler Query API is used to retrieve provenance information and to display provenance graphs of workflow execution.

### 2.2.3 WEP Generation Layer

The WEP generation layer is responsible for generating a WEP according to a scientific workflow design as shown in Figure 4. This layer contains three processes, i.e. workflow refactoring, workflow parallelization and optimization.

The workflow refactoring module refines the workflow structure for WEP generation. For instance, Ogasawara *et al.* [83, 84] take advantage of a workflow algebra to generate equivalent expressions, which are transformed into WEPs to be optimized. When a scientific workflow representation is given, it is generally not adapted for an execution environment or a SWfMS. Through workflow refactoring, a SWfMS can transform the workflow into a simpler one, e.g. by

removing redundant or useless activities, and partition it into several pieces, called fragments (by analogy with program fragments), to be executed separately by different nodes or sites. Thus, workflow partitioning is the process of decomposing a workflow into (connected) workflow fragments to yield distributed or parallel processing. A workflow fragment (or fragment for short) can be defined as a subset of activities and data dependencies of the original workflow (see [83] for a formal definition). Note that the term workflow fragment is different from the term sub-workflow, although they are sometimes confused. However, the term sub-workflow is used to refer to the relative position of a workflow in a workflow composition hierarchy [109]. Workflow partitioning is addressed in [15] for multiple execution sites (computer clusters explained in Section 2.2.5) with storage constraints. A method is proposed to partition a big workflow into small fragments, which can be executed in an execution site with moderate storage resources. In addition, Deelman *et al.* [34] propose an approach to remove workflow activities for workflow refactoring. This approach reduces redundant computational activities based on the availability of the intermediate data produced by previous execution. Tanaka and Tatebe [104] use a Multi-Constraint Graph Partitioning (MCGP) algorithm [66] to partition a workflow into fragments, which has equal weight value in each dimension while the weight of the edges crossing between fragments is minimized. In this method, each activity is defined as a vector of multiple values and each dependency between different activities has a value. This method balances the activities in each fragment while minimizing associated edges between different fragments. Moreover, workflow refactoring can also reduce workflow structure complexity. Cohen-Boulakia *et al.* [21] present a method to automatically detect over-complicated structures and replace them with easier equivalent structures to reduce workflow structure complexity.

Workflow parallelization exploits different types of parallelism to generate concrete executable tasks for the WEP. It encapsulates the related data, i.e. input, instruction and parameter data, into a task. After this operation, an activity may correspond to several tasks that are executed in parallel. Swift [123], Pegasus [34], Chiron [84] and some other SWfMSs can achieve workflow parallelization using MPI (or an MPI-like language) or a middleware within their execution engine. Since they have full control over the parallel workflow execution, these SWfMSs can leverage parallelism at different levels and yield the maximum level of performance. Some other SWfMSs outsource parallelization and workflow scheduling (see Section 2.2.4) to external execution tools, e.g. web services or Hadoop MapReduce systems. These SWfMSs can achieve activity parallelism but data parallelism (see Section 3.1.1) is generally realized in the external execution tools. The SWfMSs that outsource parallelization to a Hadoop MapReduce system adapt a data analysis process to a MapReduce workflow, composed of Map and Reduce activities. These SWfMSs generate corresponding MapReduce tasks and submit the tasks to the MapReduce system. Wang *et al.* [112] propose an architecture that combines the Kepler workflow engine with the Hadoop MapReduce framework to support the execution of MapReduce workflows. Delegating parallelization and parallel execution to an external engine makes it easy for the SWfMS to deal with very large data-intensive tasks. However, this approach is not as efficient as direct support of parallelism in the SWfMS. In particular, it makes the SWfMS loose control over the entire workflow execution, so that important optimizations, e.g. careful placement of intermediate data exchanged between tasks, cannot be realized. Furthermore, provenance management becomes almost impossible as the external tools typically do not support provenance.

Workflow optimization captures the results of workflow refactoring and workflow parallelization and inserts additional instructions for workflow scheduling to generate a WEP. The additional instructions describe multiple objectives for workflow execution, such as minimizing execution time, meeting security restrictions and reducing resource cost. The multiple objectives are mainly attained by adjusting workflow scheduling at the WEP execution layer. Having an algebra and dataflow-oriented execution engine opens up interesting opportunities for optimiza-

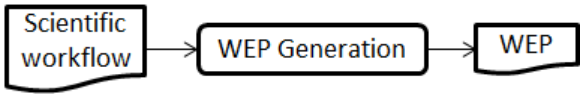



Figure 4: **WEP generation.**

tion [37, 18]. For example, it allows for user interference on the execution plan, even during the execution.

### 2.2.4 WEP Execution Layer

The WEP execution is managed at the WEP execution layer. This layer handles workflow scheduling, task execution and fault-tolerance.

Through workflow scheduling, a SWfMS produces a Scheduling Plan (SP), which aims at making good use of computing resources and preventing execution stalling [11]. A SWfMS can schedule workflow fragments, bags of tasks or individual tasks into an execution site (computer clusters explained in Section 2.2.5) or a computing node according to different task scheduling methods. Some SWfMSs distribute bags of tasks into computing nodes to reduce the scheduling overhead so that the execution time can be reduced. A bag of tasks contains several tasks to be executed in the same computer. If the execution environment contains several independent execution sites, a SWfMS can also schedule a fragment at an execution site to reduce scheduling complexity [15]. The scheduling methods are presented in Section 3.2. Some SWfMSs outsource workflow scheduling to external tools (see Section 2.2.3). Even though these SWfMSs can achieve parallelism at the task level, they cannot optimize SPs in external tools, which are generally not data-flow aware, according to the entire structure of the workflow [37].

During task execution, the input data is transferred to the computing nodes and the output data is produced. Generally, the provenance data is also generated at this time. SWfMSs can execute tasks either directly in their execution engine (e.g. Kepler, Galaxy, Pegasus, Chiron) or using an external tool (e.g. web service, MapReduce system).

The workflow fault tolerance mechanism deals with failures or errors of task execution and guarantees the availability and reliability of workflow execution. According to Ganga and Karthik [44], fault-tolerance techniques can be classified into proactive and reactive. Proactive fault tolerance avoids faults and errors by predicting the failure and proactively replacing the suspected components from other working components. Reactive fault-tolerance reduces the effect of failures after perceiving failures, using check pointing/restart, replication and task resubmission techniques. Ganga and Karthik [44] propose a task replication technique based on the idea that a replication of size $r$ can tolerate $r$-1 failed tasks while keeping the impact on the execution time minimal. Costa *et al.* [22] introduce heuristics based on real-time provenance data for detecting task execution failure and re-executing failed tasks. This heuristic re-executes failed tasks during workflow execution using extra computing resources in the cloud to reduce bad influences on the workflow execution from the task failures.

### 2.2.5 Infrastructure Layer

This layer offers an interface between SWfMSs and physical computing and storage resources through an infrastructure interface. The limitations of computing and storage resources of one computer force SWfMS users to use multiple computers in a cluster, grid or cloud infrastructure for workflow execution.

A computer cluster consists of a set of interconnected computing nodes (i.e. computers that can be considered as a single system) [19]. Cluster computing provides a paradigm of parallel computing for high performance and availability. A computer cluster, or cluster for short, is usually composed of homogenous physical computers interconnected by a high speed network, e.g. Fast Ethernet or Infiniband, but with the development of grid computing and the emergence of cloud computing, a cluster can consist of computer nodes in the grid or virtual machines in the cloud. A *virtual machine* (*VM*) is an emulator of a computer, which can be viewed as a computing node in a network. Cluster users can rely on message passing protocols, such as Message Passing Interface (MPI) [103], for parallel execution.

According to Foster and Kesselman [41], grid computing is a hardware and software infrastructure that manages distributed computers to provide good quality of service through standard protocols and interfaces with a decentralized control mechanism. Grid computing federates geographical distributed sites that are composed of diverse clusters through complex coordinating mechanisms to serve as a global system. Compared to computer cluster, grid computing gathers heterogeneous computer resources to provide more flexible services to diverse users by inner resource allocation mechanisms.

Cloud computing groups the computing, storage and network resources of one or several organizations to provide infrastructure, platform and software services through virtualization techniques, with the illusion that resources are unlimited. Although often confused, there are two main differences between cloud computing and grid computing. One is that cloud computing uses virtualization techniques to provide scalable services that are independent of physical infrastructures. The other one is that cloud computing not only provides infrastructure services such as computing resources or storage resources but also provides platform and software services. In the cloud, we can configure and use a computer cluster composed of VMs.

The operational layer is also in charge of provisioning, which can be static or dynamic. Static provisioning can provide unchangeable resources for SWfMSs during workflow execution while dynamic provisioning can add or remove resources for SWfMSs at runtime. Based on the types of resources, provisioning can be classified into computing provisioning and storage provisioning. Computing provisioning means offering computing nodes to SWfMSs while storage provisioning means providing storage resources for data caching or data persistence. However, most SWfMSs are just adapted to static computing and storage provisioning.

The data storage module generally exploit database management systems and file systems to manage all the data during workflow execution. Some SWfMSs such as Taverna put intermediate data and output data in a database. As proposed in [124], some SWfMSs such as Pegasus and Chiron utilize a shared-disk file system. Some SWfMSs such as Kepler [112] can exploit distributed file systems. Some SWfMSs such as Pegasus can directly take advantage of the local file systems in each computing node. Generally, the file systems and the database management systems take advantage of computing nodes and storage resources provided by the provisioning module. In Section 4.2.1 we will discuss file systems in detail.

## 2.3 Techniques for Data-intensive Scientific Workflows

Because they deal with *big data*, data-intensive scientific workflows have some features that make them more complicated to handle, compared with traditional scientific workflows. From the existing solutions, we can observe three main features, which we briefly discuss.

The first feature is the diversity of data sources and data formats in data-intensive scientific workflows. The data can consist of the input data stored in a shared-disk file system and the intermediate data stored in a database, etc. The data of various data sources differ in data transfer rate, data processing efficiency and data transfer time. These differences have a strong

influence on the workflow design and workflow execution. However, the workflow representation method composed of activities and dependencies for general workflows can only depict different computational components and the data dependencies among them. Thus, the data-intensive workflow representation should be adapted to be able to depict the diverse types of data.

The second feature is that moving some computational instruction data to where the input data is can be more efficient than moving the input data. This is true when the input data sets are very big while the corresponding instruction data is small. However, moving program codes across computing nodes is not always possible, for instance, because of proprietary rights or runtime compatibility.

The third feature is that not all the data needs to be kept all along the workflow execution. In particular, given fixed constraints on storage capacity allocated to the workflow execution, the intermediate data may be too big to be kept. Thus, it is important to discover and keep only the necessary data, to remove redundant data and to compress the data that is not used frequently.

There have been several studies that propose techniques for data-intensive workflow representation, data processing and redundant data removing, which we discuss below.

Albrecht *et al.* [3] propose a makeflow method for representing and running a data-intensive workflow. In their system, the input data of each activity should be explicitly specified for workflow representation or the workflow description will be regarded as incorrect. An example is shown in Figure 5 with a BLAST workflow (from bioinformatics) that has four types of activities. The first activity takes input data and splits the data into several files. The second type of (BLAST) activities searches for similarities between a short query sequence and a set of DNA or amino acid sequences [68]. The third type of activities (cat) regroups the output and errors of BLAST activities into two files. The last activity (finish) generates the final results. In Figure 5, the input data and the intermediate data are represented explicitly for further workflow textual description and execution.

Deng *et al.* [35] propose a task duplication approach in SWfMS for scheduling tasks onto multiple data centers. They schedule the tasks by comparing the task computational time and output data transmission time. For instance, let us consider two data centers as depicted in Figure 6. Tasks $t_1$ and $t_3$ and the corresponding input data $d_1$, $d_3$ are located at data center $dc_1$. Task $t_2$ at data center $dc_2$ needs to take the output of task $t_3$ as input data $d_2$. We note as $T_1$ the time to transfer the data $d_2$ from data center $dc_1$ to data center $dc_2$. We note as $T_2$, the sum of the time to transfer the input data $d_3$ from data center $dc_1$ to data center $dc_2$ and the time to execute task $t_3$. If time $T_1$ is longer than time $T_2$, the SWfMS will duplicate task $t_3$ from data center $dc_1$ to data center $dc_2$ to reduce execution time as shown in Figure 6 (b). If not, the SWfMS executes the tasks as they are and transfers the output of task $t_3$ to data center $dc_2$ as shown in Figure 6 (a). Furthermore, Raicu *et al.* [96] propose a data-aware scheduling method for data-intensive application scheduling. This approach schedules the tasks according to data location and available execution computing resources.

Yuan *et al.* [121] build an Intermediate data Dependency Graph (IDG) from data provenance of workflow execution. Based on IDG, a novel intermediate data storage strategy is developed to store the most appropriate intermediate datasets instead of all the intermediate data to reduce the storage cost during execution. Ramakrishnan *et al.* [97] propose an approach for scheduling data-intensive workflows onto storage-constrained distributed resources. This approach minimizes the amount of data by removing needless data and scheduling workflow tasks according to the storage capacity on individual resources.

There are some other techniques for data-intensive workflows, in particular, algebraic optimization and data transfer optimization. Dias *et al.* [37] discuss several performance advantages of having an algebra and dataflow-oriented execution engine for data-intensive applications.

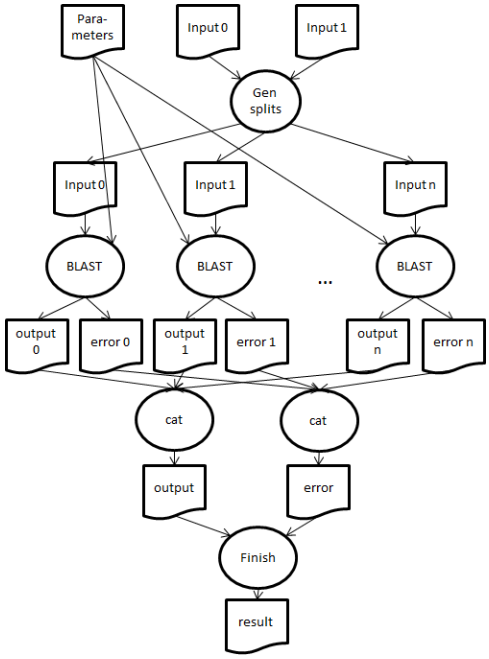



Figure 5: **BLAST workflow [3].**

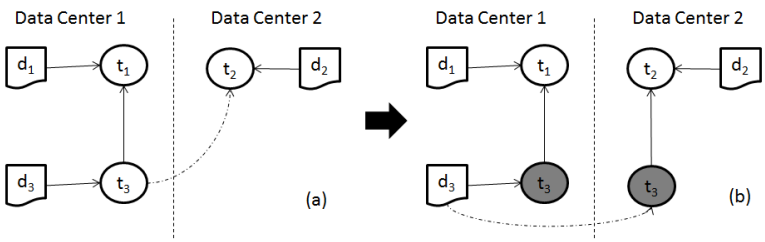



Figure 6: **Task duplication [35].**

They argue that current main approach that statically generates a WEP or an execution plan for Hadoop leaves no room for dynamic runtime changes. They propose that dataflow-based data-intensive workflows can be executed by algebraic SWfMS, such as Chiron and Scicumulus, with efficient algebraic optimizations. Moreover, we can take advantage of the former data transfer orders or current data location to control data transfer for reducing the makespan of data-intensive workflow execution. Chervenak *et al.* [17] describe a policy service that provides advice on data transfer orders and parameters based on ongoing and recent data transfers and current allocation of resources for data staging.

# 3 Parallel Execution in SWfMSs

Because of the large scale of experiments and large input or intermediate datasets, SWfMSs rely on the parallel execution of scientific workflows onto multiple computers. Workflow parallelization is the process of transforming and optimizing a (sequential) workflow into a parallel WEP. WEP allows the SWfMS to execute the workflow in parallel in a number of computing nodes, e.g. in a cluster. It is similar to the concept of Query Execution Plan (QEP) in distributed database systems [91].

This section introduces the basic techniques for the parallel execution of workflows in SWfMSs: workflow parallelization techniques; scientific workflow scheduling algorithms; and a comparative analysis of the existing solutions. The section ends with concluding remarks.

## 3.1 Workflow Parallelism

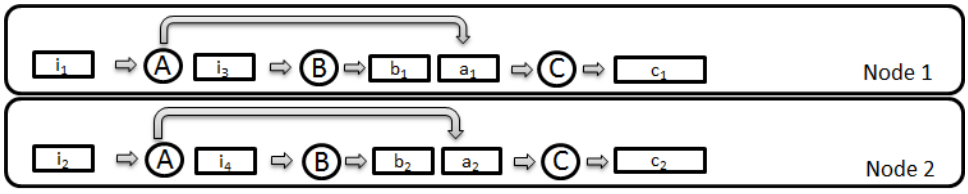
Workflow parallelization identifies the tasks that can be executed in parallel in the WEP. Similar to parallel query processing [91], whereby a QEP can be parallelized based on data and operator dependencies, there are three different types of parallelism: data parallelism, independent parallelism and pipeline parallelism. Data parallelism deals with the parallelism within an activity while independent parallelism and pipeline parallelism handle the parallelism between different activities. We explain these different types of parallelism, including their combination in hybrid parallelism, with the example shown in Figure 7.

According to the dependencies defined in a scientific workflow, different parallelization techniques can result in various execution plans. Some parameters can be used to evaluate the efficiency of each technique. An important parameter of parallelization is the degree of parallelism, which is defined as the number of concurrently running computing nodes or threads at any given time and that can vary for a given workflow depending on the type of parallelism [12].

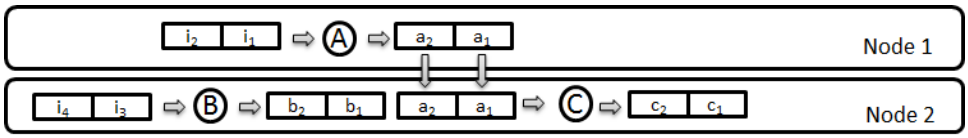
### 3.1.1 Data Parallelism

Data parallelism, similar to intra-operator parallelism in [91], is obtained by having multiple tasks performing the same activity, each on a different data chunks. As shown in Figure 7(b), data parallelism happens when the input data of an activity can be partitioned into different chunks and each chunk is processed independently by a task in a different computing node or processor. As the input data needs be partitioned, e.g. by a partitioning task, the activity result is also partitioned. Thus, the partitioned output data can be the base for data parallelism for the next activities. However, to combine the different results to produce a single result, e.g. the final result to be delivered to the user, requires special processing, e.g. by having all the tasks writing to a shared disk or sending their results to a task that produces the single result.

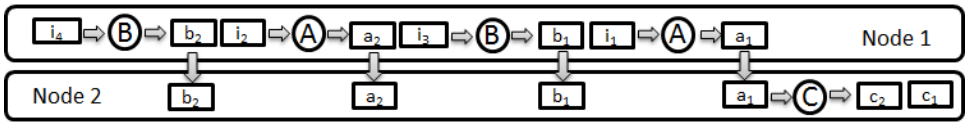

(a) **Sequential execution in one computing node.** Activity $B$ starts execution after the execution of activity $A$ and activity $C$ starts execution after the execution of activity $B$. All the execution is realized in one computing node.



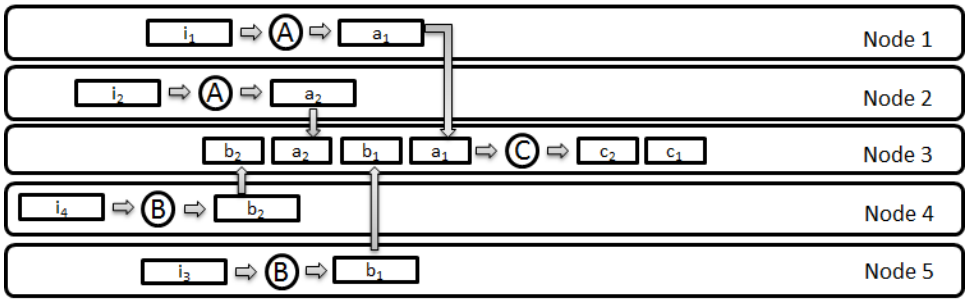

(b) **Data parallelism.** The execution of activities $A$, $B$, $C$ is performed in two computing nodes simultaneously. Each computing node processes a data chunk.




(c) **Independent parallelism.** The execution of activities $A$ and $B$ is performed in two computing nodes simultaneously. Activity $C$ begins execution after the execution of activities $A$ and $B$.




(d) **Pipeline parallelism.** Activity $C$ starts execution once a data chunk is ready. When activities $A$ and $B$ are processing the second part of data ($i_2$, $i_4$), activity C can process the output data of the first part ($a_1$, $b_1$) at the same time.




(e) **Hybrid parallelism.** Activity $A$ is executed through data parallelism at nodes 1 and 2. Activity $B$ is executed through data parallelism at nodes 4 and 5. Activities $A$ and $B$ are also executed through independent parallelism. Activities $A$ and $C$, respectively $B$ and $C$, are executed through pipeline parallelism between nodes (1, 2) and 3, respectively nodes (4, 5) and 3.

Figure 7: **Different types of parallelism.** Circles represent activities. There are three activities: $A$, $B$ and $C$. $C$ processes the output data produced by $A$ and $B$. Rectangles represent data chunks. "$i_1$" stands for the first part of input data. "$a_1$" stands for the output data corresponding to the first part of input data after being processed by activity $A$.

### 3.1.2 Independent Parallelism

Different activities of a workflow can be executed in parallel over several computing nodes. Two activities can be either independent, i.e. the execution of any activity does not depend on the output data of the other one, or dependent, i.e. there is a data dependency between them. Independent parallelism exploits independent activities while pipeline parallelism (see next subsection) deals with dependent activities.

Independent parallelism is achieved by having tasks of different independent activities executed simultaneously. As shown in Figure 7(c), independent parallelism occurs when a scientific workflow has more than one independent part in the workflow graph and the activities in each part have no data dependencies with those in another part. To achieve independent parallelism, a SWfMS should identify activities that can be executed in parallel. SWfMSs can partition the workflow into independent parts (or workflow fragments) of activities to achieve independent parallelism.

### 3.1.3 Pipeline Parallelism

With pipeline parallelism (see Figure 7(d)), several dependent activities with a producer-consumer relationship are executed in parallel by different tasks. One part of the output data of one activity is consumed directly by the next dependent activities in a pipeline fashion. The advantage of pipeline execution is that the result of the producer activity does not need to be entirely materialized. Instead, parts of data can be consumed as soon as they are ready, thus saving memory and disk accesses.

### 3.1.4 Hybrid Parallelism

As shown in Figure 7(e), the three basic types of parallelism can be combined to achieve higher degrees of parallelism. A SWfMS can first perform data parallelism within each activity. Then, it can partition the workflow into independent parts or workflow fragments for independent activities, e.g. with each part or fragment for execution in a different computing node. Finally, pipeline parallelism can be applied for executing dependent activities in parallel. In addition, the parallelism strategies may also be changed at runtime, according to the parallel computing environment behavior [26]. For the activities that process output data produced by more than one activity, the data is generally merged for the follow-up activity. This merging operation can also be found in the shuffle phase of the MapReduce program execution. By combining these mechanisms, the degree of parallelism can be maximized at different execution layers.

## 3.2 Workflow Scheduling

Workflow scheduling is a process of allocating concrete tasks to computing resources (i.e. computing nodes) to be executed during workflow execution [12]. The goal is to get an efficient Scheduling Plan (SP) that minimizes a function based on resource utilization, workflow execution cost and makespan. Since a SWfMS can schedule bags of tasks or individual tasks, there may be a task clustering phase to generate task bags. Moreover, scheduling methods can be static, dynamic or hybrid.

The SWfMSs that schedule workflow tasks without external tools choose computing nodes to execute tasks without constraints. The SWfMSs that outsource workflow parallelization or workflow scheduling may relay on external tools to schedule tasks. The following scheduling methods focus on the SWfMSs that manage workflow scheduling by themselves.

### 3.2.1 Task Clustering

A SWfMS can schedule bags of tasks to a computing nodes or multiple computing nodes. Several studies have been done for generating bags of tasks. Deng *et al.* [35] present a clustering method for efficient scientific workflow execution. They use a k-means clustering method to group the tasks into several task bags according to different dependencies: data-data dependency, task-task dependency, and task-data dependency. These three types of dependencies are used to measure the correlations between datasets and tasks in a workflow. W. Chen *et al.* [16] present a balanced task clustering approach for scientific workflow execution. They cluster the tasks by balancing total execution of each bag of task.

### 3.2.2 Static Scheduling

Static scheduling generates a SP that allocates all the executable tasks to computing nodes before execution and the SWfMS strictly abides the SP during the whole scientific workflow execution [12]. Because it is before execution, static scheduling yields little overhead at runtime. It is efficient if the SWfMS can predict the execution load of each task accurately, when the execution environment varies little during the workflow execution, and when the SWfMS has enough information about the computing and storage capabilities of the corresponding computers. However, when the execution environment experiences dynamically changes, it is very difficult to achieve load balance. The static task scheduling algorithms have two kinds of processor selection methods [106]: heuristic-based and guided random search based. The heuristic-based method schedules tasks according to a predefined rule while the random search based method schedules tasks randomly. Static task scheduling algorithms can also be classified between task-based and workflow-based [10]. The task-based method directly schedules tasks into computing nodes while the workflow-based method schedules a workflow fragment into computing nodes. Since the workflow-based method transfers the data with less overhead compared to the task-based method, it is better for data-intensive applications.

Topcuouglu *et al.* [106] propose two static scheduling algorithms: Heterogeneous Earliest-Finish-Time (HEFT) and Critical-Path-on-a-Processor (CPOP). Both algorithms contain a task prioritizing phase and a processor selection phase. The task prioritizing phase is for ranking tasks while the processor selection phase is for scheduling a selected task on a "best" computing node, which minimizes the total execution time. We note the average computation cost of a task as *CPC* and the average communication cost of the current task to another task as *CMC*. The $rank_u$ is the rank that is based on *CPC* and *CMC* that represents the communication from the current task to a succeed task. The $rank_d$ indicates the rank based on *CPC* and *CMC* consisting of the communication from a preceding task to the current task. In the task prioritizing phase, HEFT ranks tasks based on $rank_u$. In the processor selection phase, HEFT selects a computing node, which finishes its current task firsts. HEFT can also insert a task in a computing node when there is idle time between the execution of two consecutive tasks. The CPOP algorithm uses a $rank_c$ that combines both $rank_u$ and $rank_d$ in the task prioritizing phase. It utilizes a critical path in the processor selection phase. A critical path is a pipeline of tasks, in which a task has no more than one input dependency and no more than one output dependency. Each task in the critical path has the highest priority value (in $rank_c$) in all the tasks that have the input data dependencies from the same parent task. CPOP chooses a computing node as a critical-path processor and schedules the tasks in the critical path to the critical-path processor. It schedules the other tasks to the other computing nodes with the same mechanism as HEFT.

### 3.2.3 Dynamic Scheduling

Dynamic scheduling produces SPs that distribute and allocate executable tasks to computing nodes during workflow execution [12]. This kind of scheduling is appropriate for scientific workflows, in which the workload of tasks is difficult to estimate, or for environments where the capabilities of the computers varies a lot during execution. Dynamic scheduling can achieve load balancing while it takes time to dynamically generate SPs during execution. The scheduling algorithms can be based on the queue techniques in a publish/subscribe model with different strategies such as First In First Out (FIFO), adaptive and so on. SWfMSs such as Swift [115], Chiron [84], and Galaxy [65] exploit dynamic scheduling algorithms.

Dynamic SPs may be generated by adapting a static scheduling method to dynamic environment. Yu and Shi [120] introduce an HEFT-based dynamic scheduling algorithm. This algorithm is suited to the situation where a scientific workflow has been executed partially before scheduling. It schedules the tasks by applying an HEFT-based algorithm according to a dynamically generated rank of tasks.

There are some original approaches to generate dynamic SPs. Maheswaran *et al.* [72] present a min-min algorithm that is designed as a batch mode scheduling of two steps. First, a list of tasks ready to be executed is created. This phase is called "task prioritizing" phase. Then, the tasks in the list are scheduled to computing nodes based on a heuristic. The heuristic maps the task $T$ to the computing node $M$ such that $T$ is the task that has minimum expected execution time in the non-mapped tasks and that $M$ is the computing node that is executing a task having minimum expected execution time in the mapped tasks. This phase is called the "resource selection" phase.

Smanchat *et al.* [102] propose an algorithm for parameter sweep scientific workflow execution. Parameter sweep execution allows executing a scientific workflow several times with different parameters. The algorithm schedules the tasks in three phases: instance generation, task and resource prioritizing and resource selection. The authors adapt the min-min algorithm in the resource selection phase to support multiple workflow execution instances.

Anglano and Canonico [7] present several knowledge-free scheduling algorithms that are able to schedule multiple bags of tasks. A knowledge-free algorithm does not require any information on the resources for scheduling. These algorithms implement different policies: First Come First Served – Exclusive (FCFS-Excl), First Come First Served – Shared (FCFS-Share), Round Robin (RR), Round Robin –No Replica First (RR-NRF) and Longest Idle. With the FCFS-Excl policy, bags of tasks are scheduled in order of arrival. Different from FCFS-Excl, FCFS-Share can allocate more than one bag of tasks to a computing node. The RR policy schedules bags of tasks in a fixed circular order while all the bags have the same probability to be scheduled. With the RR-NRF policy, a bag of tasks that does not have any task executed will be given priority. In this policy, all the bags of tasks have at least a task running. The longest idle policy tries to reduce waiting time by giving preference to the bag of tasks hosting the task that exhibits the largest waiting time. The paper shows that the FCFS-based policy performs better for small task granularity scheduling.

### 3.2.4 Hybrid Scheduling

Both of static and dynamic scheduling have their own advantages and they can be combined as a hybrid scheduling method to achieve better performance than just using one or the other. For example, a SWfMS might schedule a part of the tasks of a workflow, e.g. those tasks for which there is enough information, using static scheduling and schedule the other part during execution with dynamic scheduling.

Oliveira *et al.* [26] propose a hybrid scheduling method with several algorithms: greedy scheduling, task grouping, task performing, and load balancing. The greedy scheduling algorithm produces static WEPs to choose the most suitable task to execute for a given idle VM based on a proposed cost model. The task grouping algorithm produces new tasks by encapsulating two or more tasks into a new one. The task performing algorithm sets up the granularity factor for each VM in the system and modifies the granularity according to the average execution time. The load balancing algorithm is a dynamic scheduling algorithm that adjusts the number of VMs and static WEP in order to meet the deadline of execution time and the budget limit.

Deng *et al.* [35] also combine static and dynamic scheduling for distributed data centers. The executable tasks are first grouped, using k-means clustering, and then scheduled following three steps: prescheduling, node adjustment and task duplication. Prescheduling generates a static SP that assigns a bag of tasks and corresponding data into an available data center while balancing the workload in each center. Node adjustment dynamically moves tasks that have more outside data transfer than inside data transfer into a more appropriate center during workflow execution. The task duplication step is presented in Section 2.3.

### 3.2.5 Scheduling Optimization Algorithms

Since there are many criteria to measure the scientific workflow execution, SWfMS users may have multiple objectives for workflow execution, such as reducing execution time, minimizing execution cost etc. Therefore, SPs should also be optimized to attain multiple objective in a given context (cluster, grid, cloud). Unlike query optimization in database, however, this optimization phase is often not explicit and mixed with the scheduling method. There are some existing scheduling optimization algorithms explained as follows.

Gu *et al.* [51] address the scheduling optimization problem of mapping distributed scientific workflows to maximize the throughput in unstable networks where nodes and links are subject to probabilistic failures. And they propose a mapping algorithm to maximize both throughput and reliability for workflow scheduling. They consider a network where the failure occurrences follow a Poisson distribution with a constant parameter. The mapping algorithm includes three algorithms: disLDP-F, Greedy disLDP-F and decentralized Greedy disLDP-F. The disLDP-F algorithm schedules the tasks by identifying and minimizing the global bottleneck time based on the rank of computational requirements of tasks. Greedy disLDP-F reduces the search complexity of disLDP-F by selecting the best node for each type of requirement of the current task and then generates a best computing node for the current task. The decentralized Greedy disLDP-F algorithm decentralizes the disLDP-F algorithm by storing all the parameters of each individual node locally and selecting the node through the communication between individual nodes instead of centralized control.

Workflow scheduling optimization in the context of data centers has been recently addressed. Oliveira *et al.* [26] propose algorithms to optimize parallel workflow execution scheduling by dynamic inserting and removing computing nodes in the cloud based on three criteria: makespan, reliability and financial cost. Coutinho *et al.* [24] propose an HGreen algorithm to reduce power cost during workflow execution. The HGreen algorithm schedules the most power-costly tasks to the most energy-efficient resources according to a rank of energy-efficient resource and a rank of power cost tasks.

## 3.3 Overview of Existing Solutions

In this section, we illustrate scientific workflow parallel execution solutions in existing SWfMSs. We start by a short presentation of parallel processing frameworks such as MapReduce. Although

they are not full-fledged SWfMS, they do share techniques in common and are often used for complex scientific data analyses, or in conjunction with SWfMS to deal with big data [112].

### 3.3.1 Parallel Processing Frameworks

Parallel processing frameworks enable the programming and execution of big data analysis applications in massively parallel computing infrastructures.

MapReduce [30] is a popular parallel processing framework for shared-nothing clusters, i.e. highly-scalable clusters with no sharing of either disk or memory among computers. MapReduce was initially developed by Google as a proprietary product to process large amounts of unstructured or semi-structured data, such as web documents and logs of web page requests, on large shared-nothing clusters of commodity nodes and produce various kinds of data such as inverted indices or URL access frequencies. Different implementations of MapReduce are now available such as Amazon MapReduce (as a cloud service) or Hadoop [113].

MapReduce enables programmers to express their computations on large data sets in a simple, functional style and hides the details of parallel data processing, load balancing and fault-tolerance. The programming model includes only two operations, *map* and *reduce*, which we can find in many functional programming languages such as Lisp and ML. The Map operation is applied to each record in the input data set to compute one or more intermediate (key,value) pairs. The Reduce operation is applied to all the values that share the same unique key in order to compute a combined result. Since they work on independent inputs, Map and Reduce can be automatically processed in parallel, on different data partitions using many computer nodes in a cluster. The input and output data is stored in a fault-tolerant distributed file system while it can be stored in a shared-disk file system or a database as well.

MapReduce execution proceeds as follows (see Figure 8). The users submit jobs composed of Map tasks or Reduce tasks to a scheduling system. When the user program calls the MapReduce function, the MapReduce library in the user program splits the input data into several pieces and makes several copies of the program distributed in the available computers. One copy of the program is the master while the others are workers that are assigned tasks by the master. The master attempts to schedule a Map task to an idle worker. The worker who is assigned a Map task processes the key/value pairs of input data chunks and puts the intermediate key/value pairs in memory. The intermediate data is written to local disk periodically after being partitioned into several regions and the location information of this data is passed to the master. The worker which has a Reduce task reads the corresponding intermediate key/value data by remote procedure calls and sorts the data through grouping the data of the same key together. Then the sorted data is passed to the user's Reduce function, which will append its output data to a final output file. When a map task or a reduce task fails, the master will reschedule it to another idle computer. When the master does not work, the users will have to retry the MapReduce operation. When all the map tasks and reduce tasks are completed, the master wakes up the user program.

Hadoop is an open source framework that supports MapReduce in a shared-nothing cluster. It uses Hadoop Distributed File System (HDFS) as storage layer (see Section 4.2). In Hadoop, MapReduce programs take input data from HDFS and put the final result and execution logs back to HDFS. Using Hadoop framework for workflow parallel execution can facilitate the implementation of SWfMSs and offer good compatibility for MapReduce programs. For instance, Wang *et al.* [112] propose an architecture to combine Kepler with Hadoop so that Kepler can represent an activity as a Map program or a Reduce program and exploit MapReduce programs to execute tasks.

*Pig* [88] is an interactive, or script-based, execution environment atop MapReduce. It sup-

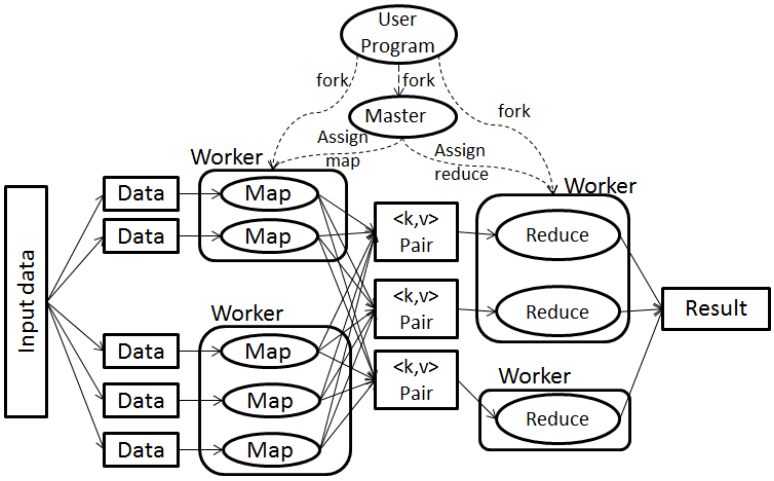



Figure 8: **MapReduce execution process.**

ports *Pig Latin*, a declarative worflow language to express large dataset analysis. PigLatin resembles SQL, with a more procedural style, and allows expressing sequences of activities that get translated in MapReduce jobs. Pig Latin can be extended using user-defined functions written in different languages like Java, Python or JavaScript. Pig programs can be run in three different ways: with a script interpreter, with a command interpreter or embedded in a Java program. Pig performs some logical optimization, by grouping activities into MapReduce jobs. For executing the activities, Pig relies on Hadoop to schedule the corresponding Map and Reduce tasks. Hadoop provides the functionality such as load-balancing and fault-tolerance. However, task scheduling and data dispatching in Hadoop is not optimized for the entire workflow.

Dryad [59] is another parallel processing framework for clusters. Similar to a scientific workflow, a Dryad job is represented as a DAG where each vertex is a program and edges represent data dependencies. Dryad handles problems of data distribution, task scheduling and fault-tolerance. The users can extend Dryad by implementing new composition operations based on two standard compositions: $A>=B$ and $A>>B$ (see Figure 9). Dryad performs greedy scheduling based on the assumption that it is the only job running on the cluster by a scheduling queue. When all of a vertex's data are ready, the execution of the corresponding program is inserted in a scheduling queue.

### 3.3.2 SWfMS

Most SWfMSs implement the five layer architecture discussed in Section 2.2. We selected eight typical SWfMSs to illustrate their techniques: Pegasus, Swift, Kepler, Taverna, Chiron, Galaxy, Triana [105] and Askalon [38]. Pegasus and Swift are two widely used SWfMSs for large-scale workflow execution in grid or cloud. Kepler, Taverna, Triana have a GUI for desktop computers. Chiron is widely used because of a powerful algebraic approach for workflow parallelization. Galaxy integrates a GUI that can be accessed through web browsers. Triana is able to use P2P services. Askalon implements both desktop and web GUI and has been adapted to work in a cloud environment.

Pegasus, Swift, Kepler and Taverna are widely used in astronomy, biology, and so on while Galaxy can only execute bioinformatics workflows. Pegasus, Swift and Chiron design and execute a workflow through a textual interface while Kepler, Taverna, Galaxy, Triana and Askalon

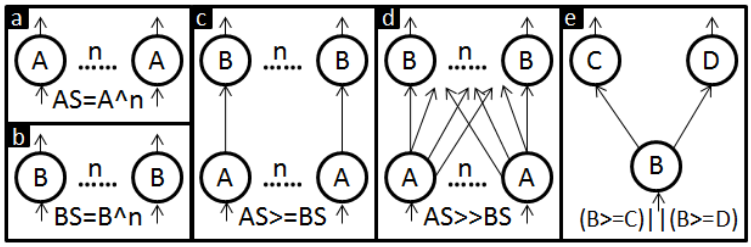



Figure 9: **Dryad operations [59].** Circles represent programs and arrows represent data dependencies. Box (a) and box (b) illustrate program cloning with the ^ operator. We note each program $P$ of type $x$ as $P_x$. The operation $AS>=BS$ in (c) means that each $P_a$ has an input data flow to each $P_b$. The operation $AS>>BS$ in (d) expresses complete bipartite composition. Box (e) shows an operation by combining the data from $P_b$ to $P_c$ and $P_d$.

integrate a GUI for workflow design. All of the eight SWfMSs support workflow specification in a DAG structure while Swift, Kepler, Chiron, Galaxy, Triana and Askalon also support workflows in a DCG structure [119]. Users can share workflow information by Taverna, Galaxy and Askalon. All of them support independent parallelism. All of them except Pegasus support dynamic scheduling and two of them (Pegasus, Kepler) support static scheduling. All the eight SWfMSs support workflow execution in both grid and cloud environments. A brief comparison of these eight SWfMSs is given in Table 1.

Table 1: **Comparison of SWfMS.** A categorization of SWfMS based on supported workflow structures, workflow information sharing, UI types, parallelism types and scheduling methods. "activity" means that this SWfMS supports both independent parallelism and pipeline parallelism.

| SWfMS | supported structures | workflow sharing | UI type | parallelism | scheduling |
|---|---|---|---|---|---|
| Pegasus | DAG | not supported | textual | data & independent | static |
| Swift | DCG | not supported | textual | activity | dynamic |
| Kepler | DCG | not supported | GUI | activity | static or dynamic |
| Taverna | DAG | supported | GUI | data & activity | dynamic |
| Chiron | DCG | not supported | textual | data & activity & hybrid | dynamic |
| Galaxy | DCG | supported | GUI (web portal) | independent | dynamic |
| Triana | DCG | not supported | GUI | data & activity | dynamic |
| Askalon | DCG | supported | GUI (desktop and web portal) | activity | dynamic & hybrid |

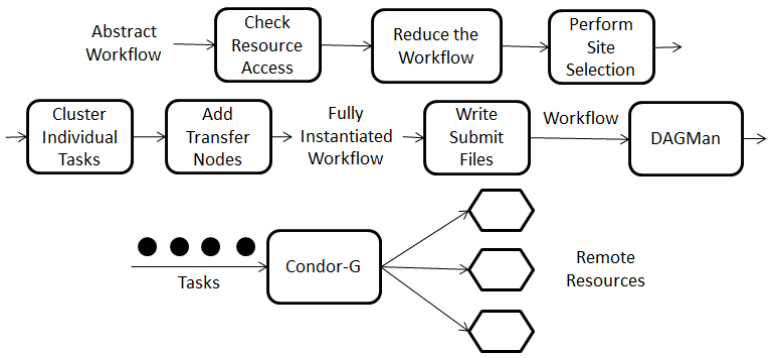



Figure 10: **Workflow execution process [34].**

**Pegasus**

Pegasus[5] is widely used in multiple disciplines such as astronomy, bioinformatics, climate modeling, earthquake science, genome analysis, etc. Pegasus has interesting features: portability on different infrastructures such as grid and cloud; optimized scheduling algorithms; good scalability; support for provenance data that can be used for debugging; data transfer support for data-intensive workflows; and fault-tolerance.

The process of executing scientific workflows in Pegasus is shown in Figure 10. In the presentation layer, Pegasus takes an abstract workflow represented as a DAG in an XML file. In the user services layer, Pegasus supports workflow monitoring through Stampede monitoring infrastructure [52, 98]. Pegasus also support provenance data gathering and querying through a Pegasus/Wings framework [67].

In the WEP generation layer, Pegasus reduces the abstract workflow by checking available intermediate data in the available computing nodes. The intermediate data can come from the previous execution of the same workflow or the execution of other workflows that contain several common activities. Pegasus performs data parallelism and independent parallelism for workflow parallelization. In the WEP execution layer, Pegasus may perform site execution based on standard algorithms (random, round-robin and min-min), data location and the significance of computation and data in the workflow execution. For example, Pegasus moves computation to the data site where big volume of data is located and it sends data to compute site if computation is significant. Pegasus clusters tasks into several bags of tasks and adds data transfer tasks to generate a fully instantiated workflow. At this point, Pegasus schedules the execution of tasks within a workflow engine such as DAGMan. Pegasus exploits static scheduling methods. In Pegasus, DAGMan sends the concrete executable tasks to Condor-G, a client tool that can manage the execution of a bag of related tasks on Grid-accessible computation nodes in the selected sites. Condor-G has a queue of tasks and it schedules a task in this queue to a computing node in the selected site once this computing node is idle [43, 69]. Pegasus handles task failures by retrying the corresponding part of workflows. Through these mechanisms, Pegasus hides the complex scheduling, optimization and data transmission of workflows from SWfMS users.

In the infrastructure layer, Pegasus is able to use computing cluster, grid and cloud to execute a scientific workflow. It exploits a shared file system for data storage and it provides static computing and storage provisioning for workflow execution.

**Swift**

Similar to Pegasus, Swift [123] has been used in multiple disciplines such as biology, astronomy,

[5]Pegasus: http://pegasus.isi.edu/

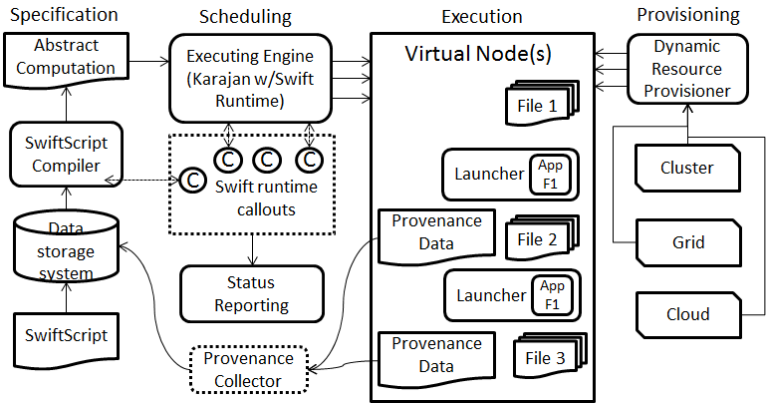



Figure 11: **Swift system architecture [123].**

economics, neuroscience, etc. Swift grew out of the GriPhyN Virtual Data System (VDS) whose objective is to express, execute, track the results of workflows through program optimization and scheduling, task management, and data management. Swift has been revised and improved its (already) large-scale performance into the Turbine system [117].

Swift executes data-intensive scientific workflows through five functional phases: program specification, scheduling, execution, provenance management and provisioning. In the presentation layer, Swift takes a workflow specification that can be described in two languages: XDTM and SwfitScript. XDTM is an interface to map the logical structure of data to physical resources. SwiftScript defines the sequential or parallel computational procedures that operate on the data defined by XDTM. In the user services layer, provenance data is available for the users.

In the WEP generation layer, the SwiftScript is compiled to an abstract computation specification. Swift generates workflow fragments for each execution site. In addition, Swift exploits independent parallelism for workflow parallelization. In the WEP execution layer, the abstract WEPs are scheduled to execution sites. The Karajan workflow execution engine is used to realize the functions such as data transfer, task submission, grid services access, task instantiation, and task schedule. Swift runtime callouts provide the information for task and data scheduling and offer status reporting, which shows the SPs. During workflow execution, provenance data is gathered by a launcher program (e.g. kickstart) to record execution information. Swift achieves fault tolerance by retrying the failed tasks and provides a restart log when the failures are permanent. The computing nodes in each execution site execute the tasks by launching corresponding applications.

In the infrastructure layer, the provisioning phase of Swift provides computing resources in a computer cluster, grid, and cloud through a dynamic resource provisioner for each execution site. The dynamic resource provisioner interacts with local or remote computing systems. Figure 11 depicts the Swift system architecture.

**Kepler**

Kepler [6, 5] is a SWfMS built upon the Ptolemy II system from the Kepler[6] project. It allows to plug in different execution models into workflows. Kepler is used in many projects of various disciplines such as oceanography[7], data management[8], and biology[9] etc. Kepler integrates a

[6]Kepler project: https://kepler-project.org/
[7]REAP project: https://kepler-project.org/users/projects-using-kepler-1/reap-project
[8]Scientific Data Management Center: https://sdm.lbl.gov/sdmcenter/
[9]Clotho project: http://www.clothocad.org/

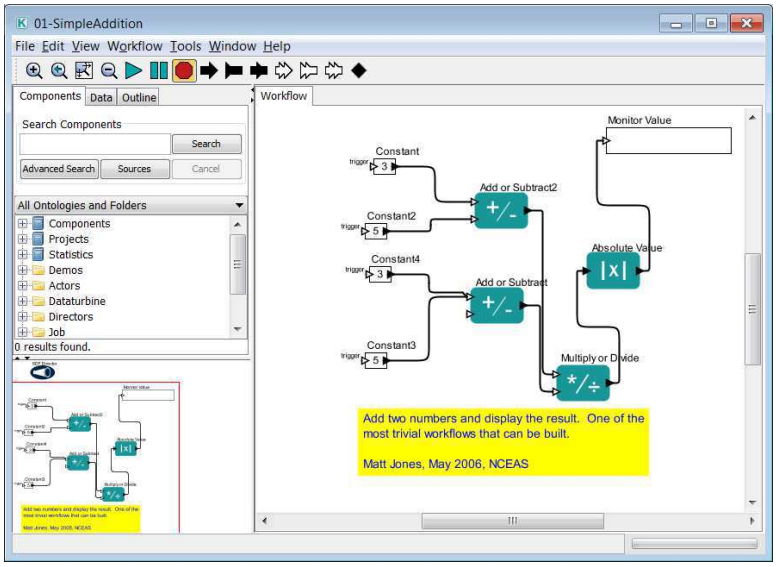



Figure 12: **Kepler workbench.**

powerful graphical workbench (shown in Figure 12). In the presentation layer, each individual reusable workflow step is implemented as an actor that can be signal processing, statistical operations, etc. Workflow activities are associated to different actors as shown in Figure 12.

In the user services layer, the provenance functionality in Kepler is realized by corresponding actors such as Provenance Recorder (PR) [4]. PR records the information of workflow execution such as context, input data, associated metadata, workflow outputs, etc.

In the WEP generation layer, the workflow is handled by a separate component named director. Kepler supports several directors and each director corresponds to a unique model of execution, which is a model of WEP. The director generates executable tasks to achieve activity parallelism (pipeline parallelism and independent parallelism).

In the WEP execution layer, Kepler exploit static or dynamic scheduling according to the director that is used during workflow execution [71, 12]. The director schedules tasks to available actors during workflow execution. The fault tolerance functionality of Kepler can be achieved by a framework that provides three complementary mechanisms. The first mechanism is a forward recovery mechanism that retries the failed tasks. The second mechanism offers a check-pointing mechanism that resumes the execution in case of a failure at the last saved state. The last one is a watchdog process that analyzes the workflow execution based on provenance data and sends an appropriate signal and possible course of action to the workflow engine to handle it. Kepler executes workflows in parallel through web services, grid-based actors or Hadoop framework. Kepler can execute workflows by using external execution environments such as SAS, Matlab, Python, Perl, C++ and R (S+) using corresponding actors.

In the infrastructure layer, Kepler can achieve data access through an OpenDBConnection actor for data in a database and an EMLDataSource actor for ecological and biological datasets. It is capable to execute scientific workflow in the grid and cloud.

**Taverna**

Taverna [77] is an open-source SWfMS from the $^{my}$Grid project to support workflow-based biological experiments. Taverna is used in multiple areas such as astronomy, bioinformatics, chemistry etc. In the presentation layer, Taverna takes a GUI for designing workflows and

showing monitoring information while it uses a textual language to represent a workflow as a DAG [119]. The workflows can be designed in Taverna installed in the user's computer or an online web server. Moreover, this GUI can be installed in an Android mobile [122]

In the user services layer, Taverna uses a state machine for the activities to achieve workflow monitoring [86]. The workflows designed through Taverna can be shared through "myExperiment" social network [116]. It gathers provenance data from local execution information and the remotely invoked web services [85].

In the WEP generation layer, Taverna automatically optimizes the workflow structure by identifying complex parts of workflow structures and simplifies them for easier design and workflow parallelization [21]. It can achieve all the three types of parallelism for workflow parallelization. Taverna links the invocation of web services and the activities and checks the availability of the needed web services for generating a WEP. In the WEP execution layer, Taverna can exploit dynamic scheduling for task scheduling. It relies on web and grid services for task execution.

In the infrastructure layer, Taverna is able to use the computing resources from grid or cloud. It also stores execution data in a database.

**Chiron**

Chiron exploits a database approach [91] to manage the parallel execution of data-intensive scientific workflows. In the presentation layer, it uses an algebraic data model to express all data as relations and represent workflow activities as algebraic expressions in the presentation layer. A relation contains sets of tuples composed of basic attributes such as integer, float, string, and file references, etc. An algebraic expression consists of algebraic activities, additional operands, operators, input relations and output relations. An algebraic activity contains a program or an SQL expression, and input and output relation schemas. An additional operand is the side information for the algebraic expression, which can be relations or a set of grouping attributes. There are six operators: Map, SplitMap, Reduce, Filter, SRQuery and MRQuery (see Section 2.2.1 for the function of each operator). In the user services layer, Chiron supports workflow monitoring, steering and gathers provenance data based on algebraic approach.

In the WEP generation layer, a scientific workflow is wholly expressed in an XML file called conceptual model. Chiron supports all types of parallelism (data parallelism, independent parallelism, pipeline parallelism, hybrid parallelism) and optimizes workflow scheduling by distinguishing between blocking activities, i.e. activities that require all their input data to proceed, and non blocking, i.e. that can be pipelined. Chiron generates concrete executable tasks for each activity and schedules the tasks of the same workflow fragments to multiple computing nodes. Chiron uses two scheduling policies, called blocking and pipeline in [37]. Let $A$ be a task that produces data consumed by a task $B$. With the blocking policy, $B$ can start only after all the data produced by $A$ are ready. Hence, there is no parallelism between $A$ and $B$. With the pipeline policy, $B$ can start as soon as some of its input data chunks are ready. Hence, there is pipeline parallelism. This pipeline parallelism is inspired by DBMS pipeline parallelism in [91]. Moreover, Chiron takes advantage of algebraic approach for workflow execution optimization to generate a WEP.

In the WEP execution layer, Chiron uses an execution module file to specify the scheduling method, database information and input data information. Chiron exploits dynamic scheduling method for task execution. Chiron gathers execution data, light domain data and provenance data all into a database into a database structured following the PROV-Wf [23] provenance model. The execution of tasks in Chiron is based on MPJ [13], an MPI-like message passing system.

In the infrastructure layer, Chiron exploits a shared-disk file system and database for data storage. It can employ the computing resources in the grid and cloud while providing just static computing and storage provisioning. Its extension, SciCumulus, is adapted to cloud environment

and can provide dynamic computing provisioning [26].

**Galaxy**

Galaxy is a web-based SWfMS for genomic research. In the presentation layer, Galaxy provides a GUI for designing scientific workflows through browsers. It can be installed in a public web server (https://usegalaxy.org/) or a private server to address specific needs.

In the user services layer, users can upload data from a user's computer or online resources and share workflow information including workflows, workflow description information, workflow input data and workflow provenance data in a public web site. Moreover, users can import workflows from "myExperiment" [116] social network [48].

In the WEP generation layer, Galaxy manages the dependencies between each activity for workflow parallelization. Galaxy achieves independent parallelism for workflow parallelization. In the WEP execution layer, Galaxy generates concrete tasks for each activity, puts the tasks in a queue to be submitted, and monitors the task status (in queue, running or completion) [65]. Through this mechanism, Galaxy exploits dynamic scheduling to dispatch executable tasks. Galaxy uses Gridway to execute tasks in the Grid. Gridway manages a task queue and the tasks in a queue are executed in an available computing node that is selected according to a greedy approach, i.e. requests are sent to all the available computing nodes while the node that has minimum response time is selected [57].

In the infrastructure layer, Galaxy can exploy computing nodes in the grid or cloud. Galaxy can exploit Globus [70] and CloudMan [2] to achieve dynamic computing and storage provisioning such as dynamic VM inserting and removing and shared-disk file system construction across computing nodes.

**Triana**

Triana [105] is a SWfMS initially developed as a data analysis tool within the GEO 600 project[10]. It provides a GUI in the presentation layer. In the user services layer, it implements the Stampede monitoring infrastructure [110] (see Section 2.2.2).

In the WEP generation layer, Triana exploits components to realize different data processing functions similar to Kepler actors. To parallel workflow execution, Triana can achieve independent parallelism, pipeline parallelism. In the WEP execution layer, Triana supports the Grid Application Toolkit (GAT) API for developing Grid-oriented components. Triana also uses the Grid Application Prototype (GAP) as an interface to interact with service-oriented networks. The GAP contains three bindings, i.e. implemented GAP, such as P2PS and JXTA to use P2P network and Web services binding to invoke Web services.

In the infrastructure layer, Triana can employ computing resources in the grid or cloud. In Section 4.4, we present how Triana has been adapted to the cloud.

**Askalon**

Askalon [38] is also a SWfMS initially designed for a grid environment. In the presentation layer, it provides a GUI, through which a scientific workflow can be modeled using Unified Modeling Language. It also exploits an XML-based language to model workflows. In the user services layer, it provides on-line workflow execution monitoring functionality through workflow execution monitoring and dynamic workflow steering to deal with exceptions in dynamic and unpredictable execution environments [95].

In the WEP generation layer, Askalon optimizes the workflow representation with loops, i.e. within DCG structures, to a DAG workflow structure. It achieves activity parallelism(independent and pipeline parallelism) for workflow parallelization and to generate a WEP. In the WEP execution layer, Askalon exploits an execution engine to provide workflow fault-tolerance at the levels of workflow, activity and control-flow. It can exploit static and hybrid scheduling, e.g. rescheduling because of unpredictable changes in the execution environment.

[10] http://www.geo600.org/

In the infrastructure layer, Askalon uses a resource manager to discover and reserve available resources and to deploy executable tasks in the grid environment. Askalon can also be deployed in a cloud environment (see Section 4.4).

## 3.4 Concluding Remarks

Data-intensive scientific workflows need to process big data, which may take a very long time with sequential execution. Parallel execution is therefore necessary to reduce execution time on parallel computers. Workflow parallel execution includes a WEP that includes parallel execution decisions, which achieves workflow parallelism. The parallel execution also schedules execution tasks to computing nodes with optimization instructions.

Workflow parallelism includes three basic types: data parallelism, independent parallelism and pipeline parallelism. Data parallelism is fine-grained parallelism within one activity and can yield a very high degree of parallelism on big datasets. Independent parallelism and pipeline parallelism exploit the parallelism between different activities. These are coarse-grained and the degree of parallelism is bound by the maximum of activities. Therefore, the highest levels of parallelism can be achieved by combining these three types of parallelism into hybrid parallelism.

Workflow scheduling is a process of allocating concrete tasks to computing node during workflow execution. Static scheduling method generates a SP prior to workflow execution and thus the workflow execution is very fast, but it makes SWfMSs difficult to achieve load balancing at a dynamically changing environment. Dynamic scheduling can better achieve load balancing but takes more time to generate SPs at run-time. Hybrid scheduling can combine the best of both static and dynamic scheduling. Workflow scheduling performs some optimization, trying to reach multiple objectives such as minimizing the makespan of workflow execution or reducing computing or storage expenses.

We observed that some SWfMSs take advantage of parallel processing frameworks such as Hadoop as lower-level tools to parallelize workflow execution and schedule tasks. This is a straightforward approach to extend a SWfMS with parallel processing capabilities. However, it lacks the capability to perform parallelization according to the entire workflow structure.

Our comparative presentation of eight SWfMSs showed that most SWfMSs do not exploit hybrid parallelism (only Chiron does) and hybrid scheduling methods (only Askalon does), which may bring the highest degrees of parallelism and good load balancing.

Although there has been much work on workflow parallelization, we believe there is a lot of room for improvement. First, input data staging needs more attention. Most SWfMSs just do this as a preprocessing step before actual workflow execution. For data-intensive scientific workflows, this step may take a very long time, for instance, to transfer several gigabytes to a computing node. Integrating this step as part of the WEP can help optimize it, based on the activities and their execution at computing nodes. Second, workflow partitioning strategies should pay attention to the computing capabilities of the resources and data to be transferred across computing nodes, as this is a major performance and cost factor, and not focus only on one constraint, e.g. storage limitation. Third, the structure of SWfMSs is generally centralized (the new version of Swift is not centralized). In this structure, a master node manages all the optimization and scheduling processes. This master node becomes a single point of failure and performance bottleneck. Distributed and P2P techniques [92] could be applied to address this problem. Fourth, although most SWfMSs are capable to produce provenance data, they lack integrated UI with provenance data which is very useful for workflow steering.

# 4 SWfMS in Multisite Cloud

The cloud, which provides virtually infinite computing and storage resources, appears as a cost-effective solution to deploy and to run data-intensive scientific workflows. For scalability and high availability, large cloud providers such as Amazon and Microsoft typically have multiple data centers located at different sites. In general, a user uses a single site, which is sufficient for most applications. However, there are important cases where scientific workflows will need to be deployed at several sites, e.g. because the data accessed by the workflow is in different research groups' databases in different sites or because the workflow execution needs more resources than those at one site. Therefore, multisite management of data-intensive scientific workflows in the cloud becomes an important problem.

This section introduces cloud computing and discusses the basic techniques for the parallel execution of scientific workflows in the cloud, including multisite management and data storage. This section ends with concluding remarks.

## 4.1 Cloud Computing

Cloud computing encompasses on demand, reliable services provided over the Internet (typically represented as a cloud) with easy access to virtually infinite computing, storage and networking resources. These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model, in which guarantees are offered by the cloud provider by means of customized Service-Level Agreements (SLAs). SLA is a part of a service contract where a service is formally defined [114]. SLA defines the quality of service provided to users by the cloud providers. One of the major differences between grid and cloud is the quality of service as Grid computing offers only best effort service. In addition, clouds provide support for pricing, accounting and SLA management.

Through very simple web interfaces and at small incremental cost, users can outsource complex tasks, such as data storage, system administration, or application deployment, to very large data centers operated by cloud providers. Thus, the complexity of managing the software/hardware infrastructure gets shifted from the users' organization to the cloud provider.

Cloud services can be divided into three broad categories: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). SaaS is the delivery of application software as a service. Hosted applications can range from simple ones such as email and calendar to complex applications such as Customer Relationship Management (CRM), data analysis or even social networks. Examples of popular SaaS are Safesforce CRM system and Microsoft Office 365.

PaaS is the delivery of a computing platform with development tools and APIs as a service. It enables developers to create and deploy custom applications directly on the cloud infrastructure, in VMs, and integrate them with applications provided as SaaS. Examples of popular PaaS are Google App Engine and Windows Azure Platform.

IaaS is the delivery of a computing infrastructure (i.e., computing, networking and storage resources) as a service. It enables customers to scale up (add more resources) or scale down (release resources) as needed (and only pay for the resources consumed). This important capability is called *elasticity* and is typically achieved through *server virtualization*, a technology that enables multiple applications to run on the same physical computer as VMs, i.e., as if they would run on distinct physical computers. Customers can then require computing instances as VMs and attach storage resources as needed. Because it is cost-effective, all cloud providers use computer clusters for their data centers, and often shared-nothing clusters with commodity

computers. Examples of popular IaaS are Amazon Elastic Compute Cloud (EC2) and Microsoft Azure.

Both SaaS, PaaS and IaaS can be useful to develop, share and execute scientific workflows components as cloud services. However, in the rest of this report, we will focus on IaaS, which will allow running existing scientific workflows in the cloud.

## 4.2 Multisite Management in the Cloud

One site in the cloud may not be big enough for providing unlimited computing and storage capability for the world. Big cloud providers such as Microsoft and Amazon typically have many geographically distributed sites. For instance, Microsoft Azure separates the world into six regions and Amazon has three sites in the USA, one site in Europe, three sites in Asia and one site in South America.

We can define a multisite cloud as a cloud composed of several sites (or data centers), each from the same or different providers and explicitly accessible to cloud users [79]. Explicitly accessible has two meanings. The first one is that each site is separately visible and directly accessible to cloud users. The second one is that the cloud users can decide to deploy their data and applications at specific sites while the cloud providers will not change the location of their data. The computing resources providers include grid computing and computer cluster providers.

In a multisite cloud environment, cloud users must take care of the location of their data, which can be difficult. A multisite cloud platform is a solution that can manage several sites (or data centers) of single or multiple cloud providers, with a uniform interface for cloud users.

BonFIRE [58] is a European Research project[11] that develops a multisite cloud platform for applications, services and systems experimentation. It adopts a federated multi-platform approach, providing interconnection and interoperation between service and networking testbeds. As an IaaS, it provides large-scale, virtualized computing, storage and networking resources with full control of the user on resource deployment. It also provides in-depth monitoring and logging of physical and virtual resources and ease of use of experimentation. BonFIRE currently comprises several (7 at the time of this writing) geographically distributed testbeds across Europe. Each testbed provides its computing, storage and network resources and can be accessed seamlessly with a single experiment descriptor through the BonFIRE API, which is based on the Open Cloud Computing Interface.

U-chupala *et al.* [108] propose a multisite cloud platform based on a Virtual Private Network (VPN) and a smart VM scheduling mechanism. It is composed of a virtual infrastructure layer, an overlay network layer and a physical resource layer. The VM containers lie in the physical resource layer. The overlay network connects all the physical resources together and enables the virtual infrastructure layer to use a cloud framework that gives the illusion of a single pool of resources. This pool can provide scalable resources to users while hiding the complexity of the physical infrastructure underneath.

A multisite cloud platform may contain modules coming from existing frameworks. Mandal *et al.* [74] have implemented the Hadoop framework in a multisite cloud through a cloud control framework that can gather computing and storage resources from multiple sites and offer a common interface for resource provisioning. They conclude that deploying Hadoop in networked clouds is not difficult but low quality of network yields poor performance.

In [107], an extension of MapReduce is proposed to deal with multisite cloud. A MapReduce application is partitioned for several sites according to the available data chunks distributed at each site. In this architecture (see Figure 13), a MetaReducer is implemented as an independent service and built on top of a pool of reducers distributed over multiple sites. It is used to generate

[11] http://www.bonfire-project.eu

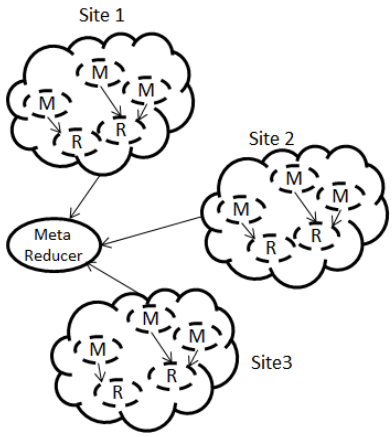



Figure 13: **Running MapReduce across a multisite cloud [107].**

the final result by aggregating all the intermediate results generated by the reducers at different sites.

## 4.3 Data Storage in the Cloud

Data storage in the cloud is critical for the performance of data-intensive scientific workflows. It can be done using different file systems. In this section, we discuss the techniques for file systems that can be used in the cloud.

### 4.3.1 File Systems

A file system is in charge of controlling how information is stored and retrieved in a computer or a computer cluster [8]. In the cloud, IaaS users need a file system that can be concurrently accessible for all the VMs. This can be achieved through a shared-disk file system or a distributed file system.

**Shared-disk file systems**
In a shared-disk file system, all the computing nodes of the cluster share some data storage that are generally remotely located. Examples of shared-disk file systems include General Parallel File System (GPFS) [100], Global File System (GFS) [94] and Network File System (NFS) [99].

A shared-disk file system is composed of data storage servers, a Storage Area Network (SAN) with fast interconnection (e.g. Infiniband or Fiber (GPFS), Channel) and is accessible to each computing node. The data storage servers offer block data level storage that is connected to each computing node by storage are network. The data in data storage servers can be read or written as in the local file system. The shared-disk file system handles the issues of concurrent access to file data, fault-tolerance at the file level and big data throughput.

Let us illustrate with General Parallel File System (GPFS), IBM's shared-disk file system. GPFS provides the behavior of a general-purpose POSIX file system running on a single computing node. GPFS's architecture (see Figure 14) consists of file system clients, a fast interconnection network and file system servers, which just serve as an access interface to the shared disks. The file system clients are the computer nodes in a cluster that need to read or write data from the

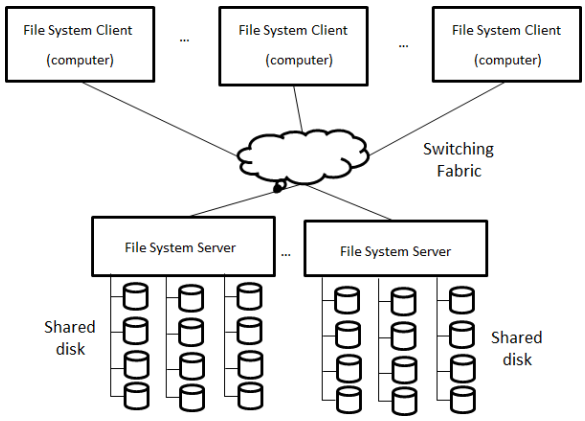



Figure 14: **GPFS architecture.**

shared disk for their installed programs. The interconnection network connects the file system clients to the shared disks through a conventional block I/O interface.

GPFS provides fault-tolerance in large-scale clusters in three situations. Upon a node failure, GPFS will restore metadata updated by the failed node to a consistent state and release lock tokens in the failed node and appoint others nodes for special roles played by the failed node. Upon a communication failure, the mechanism for one node lost is handled as the node failures while a network equipment failure causes a network partition. In the case of partition, the nodes in the partition that has the highest number of nodes have access to the shared disks. GPFS uses data replication across multiple disks to deal with disk failures.

Cloud users can deploy a shared-disk file system by installing the corresponding frameworks (e.g. GPFS framework) in the VMs with the cloud storage resources such as Microsoft Blob Storage and Amazon Elastic Block Store (EBS). Alternatively, cloud users can mount Amazon Simple Storage Service (S3) into all the Linux-based VMs to realize the functionality that all the VMs can have access to the same storage resource, as with a shared-disk file system.

**Distributed file systems**

A distributed file system stores data directly in the file system that is constructed by gathering storage space in each computing node in a shared-nothing architecture. The distributed file system integrates solutions for load balancing among computing nodes, fault-tolerance and concurrent access. Files must be partitionned into chunks, e.g. through a hash function on records' keys, and the chunks are distributed among computing nodes. Different from the shared-disk file system, computing nodes have to load the data chunks from the distributed file system to the local system before local processing.

Let us illustrate with Google File System (GFS) [45], which had a major impact on cloud data management. For instance, Hadoop Distributed File System (HDFS) is an open source framework based on GFS. GFS is designed for a shared-nothing cluster made of commodity computers, and applications with frequent read operations while write operations mainly consist of appending new data. GFS is composed of a single GFS master node and multiple GFS chunk servers. The GFS master maintains all the file system metadata while GFS chunk servers store all the real data. The master can send instruction information to the chunk servers while the chunk server can send chunk server status information to the master. A GFS client can get the data location information from the file namespace of the GFS master. Then it can write data to

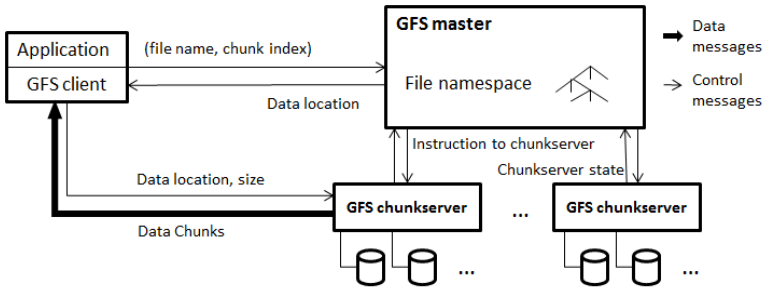



Figure 15: **GFS architecture [45].**

the GFS chunk servers at this data location or get the data chunks from a corresponding GFS chunk server according to the data location information and required data size. This mechanism is shown in Figure 15. GFS also provides snapshot support, garbage collection, fault-tolerance and diagnosis. For high availability, GFS supports master replication and data chunk replication.

BlobSeer [80] is another distributed file system optimized for Binary Large OBjects (BLOBs). The architecture of BlobSeer is shown in Figure 16. Data providers physically store the data in the storage resources (data providers) while physical storage resources can be inserted or removed dynamically in the data providers. The provider manager tracks the information about the storage resources and schedules the placement of newly generated data. All the stored data has a version. Metadata providers store the metadata for identifying data chunks that make up a snapshot version. The version manager assigns new snapshot version numbers to writers and appenders and reveals new snapshots to readers. The write operation is performed in parallel on data chunks and creates a new version of the data. Because of data versioning, read and write operations can be asynchronous and thus improve the read and write throughput. The client can get the data location of the required files corresponding to the file name and the required version when the required version is equal or inferior to the latest snapshot version. Then it can write data to the data providers or get the corresponding data chunks from the data providers by the data location and desired data size. BlobSeer also provides fault-tolerance through replication, consistency semantics and scalability based on several versioning mechanisms. Nicolae *et al.* [80] made a first performance comparison of Blobster with HDFS, which shows important improvements in read and write thoughput, because of versioning.

Cloud users can deploy a distributed file system by installing corresponding frameworks (e.g. HDFS) of the aforementioned systems in available VMs to gather storage resources in each VM for executing applications in the Cloud.

## 4.4 Scientific Workflow Execution in the Cloud

The cloud has some useful features to execute scientific workflows. In particular, the quality of service guaranteed by SLA can yield more stable performance. Juve *et al.* [64] compare the performance of an astronomy application with the Pegasus SWfMS in the grid, the commercial cloud and the academic cloud. They conclude that the performance is the least stable in grid and more stable in commercial cloud than academic cloud. Furthermore, the execution model for scientific workflows in the grid can be reused in the cloud [32]. Finally, the scalable resource provisioning can help the SWfMS to meet the execution time and budget limit for scientific workflows [26]. For instance, Oliveira *et al.* [26] take advantage of Amazon EC2 for dynamic elastic computing provisioning to meet the time requirement of executing scientific workflows

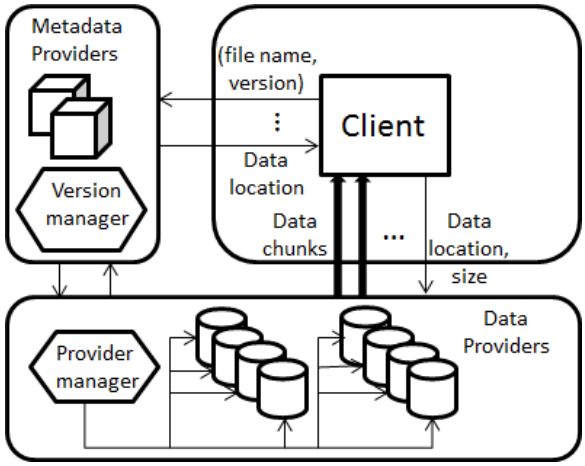



Figure 16: **BlobSeer Architecture [80].**

through SciCumulus. Afgan *et al.* [2] propose CloudMan that permits Galaxy to make use of Amazon EC2 and EBS for computing and storage provisioning for scientific workflow execution.

Because of these features, SWfMSs need be adapted to work in a cloud. SWfMSs can be directly installed in the VMs and exploit services deployed in the cloud [62, 116]. Some SWfMSs are made compatible with the cloud environment through a middleware such as Coasters [53] in Swift, Kepler EC2 actors [111] for Kepler, CloudMan [2] for Galaxy and RabbitMQ12 for Triana[12] SWfMS. These tools can provide computing or storage provisioning for scientific workflow execution or communication between VMs. However, they cannot take advantage of the dynamic provisioning features of the cloud.

Some other SWfMSs are optimized for the cloud by supporting dynamic resource provisioning, task scheduling under budget and time limits such as Pegasus [73, 78] with Wrangler [63] (a dynamic provisioning system in the cloud) and Askalon [90, 89, 39] SWfMS. Chiron is adapted to the cloud through its extension, Scicumulus [27, 28]. The architecture of Scicumulus contains three layers and four corresponding tiers: desktop layer for client tier, distribution layer for distribution tier, execution layer for execution tier and data tier. The desktop layer is to compose and execute workflows. The distribution layer is responsible for parallel execution of workflow activities in the cloud. The execution layer manages workflow activity execution in VM instances. Finally, the data tier manages the related data during workflow execution. Scicumulus exploits hybrid scheduling approaches with dynamic computing provisioning support. Furthermore, Scicumulus uses services such as SciDim [29] to determine an initial virtual cluster size through a multi-objective cost function and provenance data under budget and time limits. Moreover, Scicumulus can be coupled with SciMultaneous, which is used to manage fault tolerance in the cloud [22].

### 4.4.1 Scientific Workflow Execution in Multisite Cloud

In some cases, a scientific workflow must be executed at multiples sites. There are two approaches to do this. The first approach is to deploy a SWfMS in a multisite cloud platform as discussed in Section 4.3. The second approach is to make a SWfMS multisite-aware and capable to utilize

[12]Triana in cloud: `http://www.trianacode.org/news.php`

computing and storage resources distributed at different sites. This is the approach we now focus on.

In a multisite cloud, we can execute a scientific workflow with or without workflow partitioning. Scientific workflow execution without workflow partitioning consists of scheduling all the tasks into all the VMs in the multiple sites directly. This centralized method makes it hard to realize load balancing, incurs much overhead for each task and makes scheduling very complicated. With workflow partitioning, a workflow is divided in workflow fragments (see Section 3.1) and each fragment is scheduled at a specific site and its tasks scheduled within the VMs at this site. This method can reduce the overhead of task scheduling, which is done in parallel at multiple sites, and realize load balancing at two levels: inter-site and intra-site. Inter-site load balancing is realized by scheduling fragments, with a global scheduler, and intra-site load balancing is realized by local task scheduling. This two-level approach makes the scheduling operation easier.

Swift and Pegasus achieve multisite execution by workflow partitioning. Swift performs workflow partitioning by generating corresponding abstract WEPs for each site [123]. Pegasus realizes workflow partitioning through several methods [15, 16]. As discussed in Section 2.2.3, Chen and Deelman [15] propose a workflow partitioning method under storage constraints at each site. This workflow partitioning method is used in a multisite environment with dynamic computing provisioning as explained in [14]. Another method is balanced task clustering [16]. The workflow is partitioned into several workflow fragments which have almost the same workload. This method can realize load balancing for homogeneous computing resources. Askalon can execute scientific workflows in a federated multisite cloud [90], i.e. a multisite cloud composed of resources from different providers. Nevertheless, it schedules tasks in computing nodes without considering the organization of computing resources, i.e. which VMs are at the same site, for optimization. This method just takes the VMs as the grid computing nodes without considering the features of multisite resources, e.g. the difference of data transfer rate, resource sharing for intra-site and inter-site, etc.

## 4.5 Conclusion and Remarks

There are important cases where scientific workflows will need to be deployed at several data centers in the cloud, either from the same or different cloud providers, thus making multisite management an important problem. Although some SWfMSs such as Swift and Pegasus provide some functionality to execute scientific workflows in the multisite environment, this is generally done by simply reusing the techniques from a grid environment or simple dynamic provisioning and scheduling mechanisms, without exploiting new data storage and data transfer capabilities provided by multisite clouds.

We believe that much more work is needed to improve the execution of data-intensive scientific workflows in a multisite cloud. First, the communication between two sites is generally achieved by having two nodes, each at one of the two sites, communicating directly, which is not efficient in a multisite cloud. For instance, selecting several nodes at one site to send or receive data to or from several nodes at another site could exploit parallel data transfer and make it more efficient. Second, the workflow partitioning algorithm should consider multisite execution, considering the computing and data transfer capabilities of the sites. Furthermore, the co-scheduling of tasks and data should be exploited. Most SWfMSs make use of file systems or database systems to store data but do not care about where the data is stored during workflow execution. We believe that the co-scheduling of tasks and data can be efficient at maximizing local data processing. Finally, SWfMSs should optimize the scheduling of workflow fragments and tasks for the architecture of computing resources and storage in multiple cloud sites.

# 5 Conclusion

In this paper, we discussed the current state of the art of data-intensive scientific workflow execution in the cloud and corresponding multisite management techniques.

First, we introduced the definitions in scientific workflow management, including scientific workflows and SWfMSs. In particular, we illustrated the representation of scientific workflows with real examples from astronomy and biology. Then, we presented in more details a five-layer functional architecture of SWfMSs and the corresponding functions. Special attention has been paid to data-intensive workflows by identifying their features and presenting the corresponding techniques.

Second, we introduced the basic techniques for the parallel execution of workflows in SWfMSs: parallelization and scheduling. We showed how different kinds of parallelism (data parallelism, independent parallelism and pipeline parallelism) can be exploited for parallelizing scientific workflows. The scheduling methods to allocate tasks to computing resources can be static or dynamic, with different trade-offs, or hybrid to combine the advantages of static and dynamic scheduling methods. Workflow scheduling may include an optimization phase to minimize a multi-objective function, in a given context (cluster, grid, cloud). However, unlike in database query optimization, this scheduling optimization phase is often not explicit and mixed with the scheduling method. To illustrate the use of these techniques, we introduced the recent parallelization frameworks such as MapReduce and gave a comparative analysis of eight popular SWfMSs (Pegasus, Swift, Kepler, Taverna, Chiron, Galaxy, Triana and Askalon).

Third, we focused on the management of SWfMSs in multisite cloud. We introduced cloud computing, and discussed the basic techniques for the parallel execution of scientific workflows in the cloud, including data storage (file systems and database management systems) and multisite management.

The current solutions for the parallel execution of SWfMSs are appropriate for static computing and storage resources in a grid environment. They have been extended to deal with more elastic resources in a cloud, but with one site only. Our analysis of the current techniques of scientific workflow parallelization and scientific workflow execution has shown that there is a lot of room for improvement. And we proposed directions of future research, which we summarize as follows:

1. Workflow representation: existing workflow representations can just present the activities and data dependencies in a workflow while they cannot represent the diversity of data formats or special data sources such as big data stored in a specific data center. New workflow representations are needed for data-intensive scientific workflows.

2. Data staging: efficient data transmission between sites is critical for data-intensive workflow execution. Existing techniques mainly focus on the mechanism that starts scientific workflow execution after gathering all the related data in a shared-disk file system at one data center, which is time consuming. New data staging methods, including caching, are needed to increase efficiency of data transmission in scientific workflow execution.

3. Architecture: the structure of SWfMSs is generally centralized, with a master node managing all the optimization and scheduling processes. This master node becomes a single point of failure and performance bottleneck. Distributed and P2P techniques could be applied to address this problem.

4. Task scheduling and data location: most SWfMSs do not take data location into consideration during task scheduling period. For data-intensive scientific workflows, a uniform

scheduling method is needed to handle task and data scheduling at the same time. Furthermore, SWfMSs should also take advantage of the organization of computing and storage resources in multiple cloud sites to schedule workflow fragments or tasks into available computing nodes.

5. Multisite: novel task and data scheduling approaches are required for utilizing resources in a multisite cloud. Furthermore, to partition a workflow into several parts based on resources in each site is also a difficult optimization problem in a multisite environment.

# References

[1] M. Abouelhoda, S. Issa, and M. Ghanem. Tavaxy: Integrating taverna and galaxy workflows with cloud computing support. *BMC Bioinformatics*, 13(1):77, 2012.

[2] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor. Galaxy cloudman: delivering cloud compute clusters. *BMC Bioinformatics*, 11(Suppl 12):S4, 2010.

[3] M. Albrecht, P. Donnelly, P. Bui, and D. Thain. 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*, pages 1:1–1:13, 2012.

[4] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In *Int. Conf. on Provenance and Annotation of Data*, pages 118–132, 2006.

[5] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *16th Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 423–424, 2004.

[6] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: Towards a Grid-Enabled system for scientific workflows. *The Workflow in Grid Systems Workshop in GGF10-The 10th Global Grid Forum*, 2004.

[7] C. Anglano and M. Canonico. Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach. In *22nd IEEE Int. Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8, 2008.

[8] K. Arkoudas, K. Zee, V. Kuncak, and M. Rinard. Verifying a file system implementation. In *6th Int. Conf. on Formal Engineering Methods (ICFEM)*, volume 3308, pages 373–390. 2004.

[9] K. Belhajjame, S. Cresswell, Y. Gil, R. Golden, P. Groth, G. Klyne, J. McCusker, S. Miles, J. Myers, and S. Sahoo. The prov data model and abstract syntax notation. `http://www.w3.org/TR/2011/WD-prov-dm-20111215/`, 2011.

[10] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *5th IEEE Int. Symposium on Cluster Computing and the Grid (CCGrid)*, pages 759–767, 2005.

[11] L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. Dynamic query scheduling in data integration systems. In *International Conference on Data Engineering (ICDE)*, pages 425–434, 2000.

[12] M. Bux and U. Leser. Parallelization in scientific workflow management systems. *The Computing Research Repository (CoRR)*, abs/1303.7195, 2013.

[13] B. Carpenter, V. Getov, G. Judd, A. Skjellum, and G. Fox. Mpj: Mpi-like message passing for java. *Concurrency and Computation: Practice and Experience*, 12(11):1019–1038, 2000.

[14] W. Chen and E. Deelman. Integration of workflow partitioning and resource provisioning. In *IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGRID)*, pages 764–768, 2012.

[15] W. Chen and E. Deelman. Partitioning and scheduling workflows across multiple sites with storage constraints. In *9th Int. Conf. on Parallel Processing and Applied Mathematics - Volume Part II*, volume 7204, pages 11–20, 2012.

[16] W. Chen, R. D. Silva, E. Deelman, and R. Sakellariou. Balanced task clustering in scientific workflows. In *IEEE 9th Int. Conf. on e-Science*, pages 188–195, 2013.

[17] A. L. Chervenak, D. E. Smith, W. Chen, and E. Deelman. Integrating policy with scientific workflow management for data-intensive applications. In *Supercomputing (SC) Companion: High Performance Computing, Networking Storage and Analysis*, pages 140–149, 2012.

[18] F. Chirigati, V. Silva, E. Ogasawara, D. de Oliveira, J. Dias, F. Porto, P. Valduriez, and M. Mattoso. Evaluating parameter sweep workflows in high performance computing. In *1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, pages 2:1–2:10, 2012.

[19] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 41(4):98–109, 2011.

[20] W. M. Coalition. Workflow management coalition terminology and glossary, 1999.

[21] S. Cohen-Boulakia, J. Chen, P. Missier, C. A. Goble, A. R. Williams, and C. Froidevaux. Distilling structure in taverna scientific workflows: a refactoring approach. *BMC Bioinformatics*, 15(S-1):S12, 2014.

[22] F. Costa, D. de Oliveira, K. Ocala, E. Ogasawara, J. Dias, and M. Mattoso. Handling failures in parallel scientific workflows using clouds. In *Supercomputing (SC) Companion: High Performance Computing, Networking Storage and Analysis*, pages 129–139, 2012.

[23] F. Costa, V. Silva, D. de Oliveira, K. A. C. S. Ocaña, E. S. Ogasawara, J. Dias, and M. Mattoso. Capturing and querying workflow runtime provenance with prov: a practical approach. In *EDBT/ICDT Workshops*, pages 282–289, 2013.

[24] F. Coutinho, L. de Carvalho, and R. Santana. A workflow scheduling algorithm for optimizing energy-efficient grid resources usage. In *IEEE 9th Int. Conf. on Dependable, Autonomic and Secure Computing (DASC)*, pages 642–649, 2011.

[25] D. Crawl, J. Wang, and I. Altintas. Provenance for mapreduce-based data-intensive workflows. In *6th Workshop on Workflows in Support of Large-scale Science*, pages 21–30, 2011.

[26] D. de Oliveira, K. A. C. S. Ocaña, F. Baião, and M. Mattoso. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *Journal of Grid Computing*, 10(3):521–552, 2012.

[27] D. de Oliveira, E. Ogasawara, F. Baião, and M. Mattoso. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *3rd Int. Conf. on Cloud Computing (CLOUD)*, pages 378–385, 2010.

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

[29] D. de Oliveira, V. Viana, E. Ogasawara, K. Ocana, and M. Mattoso. Dimensioning the virtual cluster for parallel scientific workflows in clouds. In *4th ACM Workshop on Scientific Cloud Computing*, pages 5–12, 2013.

[30] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150, 2004.

[31] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[32] E. Deelman, G. Juve, and G. B. Berriman. Using clouds for science, is it just kicking the can down the road? In *Cloud Computing and Services Science (CLOSER), 2nd Int. Conf. on Cloud Computing and Services Science*, pages 127–134, 2012.

[33] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The montage example. In *ACM/IEEE Conf. on High Performance Computing*, pages 1–12, 2008.

[34] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[35] K. Deng, L. Kong, J. Song, K. Ren, and D. Yuan. A weighted k-means clustering based co-scheduling strategy towards efficient execution of scientific workflows in collaborative cloud environments. In *IEEE 9th Int. Conf. on Dependable, Autonomic and Secure Computing (DASC)*, pages 547–554, 2011.

[36] J. Dias, D. de Oliveira, M. Mattoso, K. A. C. S. Ocana, and E. Ogasawara. Discovering drug targets for neglected diseases using a pharmacophylogenomic cloud workflow. In *IEEE 8th Int. Conf. on E-Science (e-Science)*, pages 1–8, 2012.

[37] J. Dias, E. S. Ogasawara, D. de Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Algebraic dataflows for big data analysis. In *IEEE Int. Conf. on Big Data*, pages 150–155, 2013.

[38] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. Truong, A. Villazon, and M. Wieczorek. Askalon: A development and grid computing environment for scientific workflows. In *Workflows for e-Science*, pages 450–471. Springer, 2007.

[39] H. M. Fard, T. Fahringer, and R. Prodan. Budget-constrained resource provisioning for scientific applications in clouds. In *IEEE 5th Int. Conf. on Cloud Computing Technology and Science (CloudCom)*, volume 1, pages 315–322, 2013.

[40] J. Felsenstein. Phylip - phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

[41] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 2003.

[42] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[43] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: a computation management agent for multi-institutional grids. In *10th IEEE Int. Symposium on High Performance Distributed Computing*, pages 55–63, 2001.

[44] K. Ganga and S. Karthik. A fault tolerent approach in scientific workflow systems based on cloud computing. In *Int. Conf. on Pattern Recognition, Informatics and Mobile Engineering (PRIME)*, pages 387–390, 2013.

[45] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *19th ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.

[46] Y. Gil, J. Kim, V. Ratnakar, and E. Deelman. Wings for pegasus: A semantic approach to creating very large scientific workflows. In *OWLED*06 Workshop on OWL: Experiences and Directions*, volume 216, 2006.

[47] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):1–13, 2010.

[48] J. Goecks, A. Nekrutenko, and J. Taylor. Lessons learned from galaxy, a web-based platform for high-throughput genomic analyses. In *IEEE Int. Conf. on E-Science, e-Science*, pages 1–6, 2012.

[49] J. A. R. Gonçalves, D. Oliveira, K. Ocaña, E. Ogasawara, and M. Mattoso. Using domain-specific data to enhance scientific workflow steering queries. In *Provenance and Annotation of Data and Processes*, volume 7525, pages 152–167. 2012.

[50] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, and M. Reiter. Conventional workflow technology for scientific simulation. In *Guide to e-Science*, pages 323–352. 2011.

[51] Y. Gu, C. Wu, X. Liu, and D. Yu. Distributed throughput optimization for large-scale scientific workflows under fault-tolerance constraint. *Journal of Grid Computing*, 11(3):361–379, 2013.

[52] D. Gunter, E. Deelman, T. Samak, C. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swany, and K. Vahi. Online workflow management and performance analysis with stampede. In *7th Int. Conf. on Network and Service Management (CNSM)*, pages 1–10, 2011.

[53] M. Hategan, J. Wozniak, and K. Maheshwari. Coasters: Uniform resource provisioning and access for clouds and grids. In *4th IEEE Int. Conf. on Utility and Cloud Computing*, pages 114–121, 2011.

[54] F. Hernández and T. Fahringer. Towards workflow sharing and reusein the askalon grid environment. In *Proceedings of Cracow Grid Workshops (CGW)*, page 111–119, 2008.

[55] S. Holl, O. Zimmermann, and M. Hofmann-Apitius. A new optimization phase for scientific workflow management systems. In *8th IEEE Int. Conf. on E-Science*, pages 1–8, 2012.

[56] F. Horta, J. Dias, K. Ocana, D. de Oliveira, E. Ogasawara, and M. Mattoso. Abstract: Using provenance to visualize data from large-scale experiments. In *Supercomputing (SC): High Performance Computing, Networking Storage and Analysis*, pages 1418–1419, 2012.

[57] E. Huedo, R. S. Montero, and I. M. Llorente. A framework for adaptive execution in grids. *Software - Practice and Experience (SPE)*, 34(7):631–651, 2004.

[58] A. C. Hume, Y. Al-Hazmi, B. Belter, K. Campowsky, L. M. Carril., G. Carrozzo, V. Engen, D. García-Pérez, J. J. Ponsatí, R. K. ubert, Y. Liang, C. Rohr, and G. Seghbroeck. Bonfire: A multi-cloud test facility for internet of services experimentation. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 44, pages 81–96. 2012.

[59] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *2nd ACM SIGOPS/EuroSys European Conf. on Computer Systems*, pages 59–72, 2007.

[60] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. A. Prince, and R. Williams. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *Int. Journal of Computational Science and Engineering*, 4(2):73–87, 2009.

[61] L. M. R. G. Jr., M. Wilde, M. Mattoso, and I. Foster. Provenance traces of the swift parallel scripting system. In *EDBT/ICDT Workshops*, pages 325–326, 2013.

[62] G. Juve and E. Deelman. Scientific workflows in the cloud. In *Grids, Clouds and Virtualization*, pages 71–91. Springer, 2011.

[63] G. Juve and E. Deelman. Wrangler: Virtual cluster provisioning for the cloud. In *20th Int. Symposium on High Performance Distributed Computing*, pages 277–278, 2011.

[64] G. Juve, M. Rynge, E. Deelman, J.-S. Vockler, and G. Berriman. Comparing futuregrid, amazon ec2, and open science grid for scientific workflows. *Computing in Science Engineering*, 15(4):20–29, 2013.

[65] K. Karuna, N. Mangala, C. Janaki, S. Shashi, and C. Subrata. Galaxy workflow integration on garuda grid. In *IEEE Int. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 194–196, 2012.

[66] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *ACM/IEEE Conf. on Supercomputing*, pages 1–13, 1998.

[67] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the wings-pegasus system. *Concurrency and Computation: Practice and Experience*, 20:587–597, 2008.

[68] I. Korf, M. Yandell, and J. A. Bedell. *BLAST - an essential guide to the basic local alignment search tool*. O'Reilly, 2003.

[69] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor-a hunter of idle workstations. In *8th Int. Conf. on Distributed Computing Systems*, pages 104–111, 1988.

[70] B. Liu, B. Sotomayor, R. Madduri, K. Chard, and I. Foster. Deploying bioinformatics workflows on clouds with galaxy and globus provision. In *Supercomputing (SC) Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*, pages 1087–1095, 2012.

[71] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. B. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[72] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *8th Heterogeneous Computing Workshop*, pages 30–, 1999.

[73] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Supercomputing (SC) Conf. on High Performance Computing Networking, Storage and Analysis*, pages 1–11, 2012.

[74] A. Mandal, Y. Xin, I. Baldine, P. Ruth, C. Heerman, J. Chase, V. Orlikowski, and A. Yumerefendi. Provisioning and evaluating multi-domain networked clouds for hadoop-based applications. In *Cloud Computing Technology and Science (CloudCom), IEEE 3rd Int. Conf. on Cloud Computing Technology and Science*, pages 690–697, 2011.

[75] M. Mattoso, K. O. na, F. Horta, J. Dias, E. Ogasawara, V. Silva, D. de Oliveira, F. Costa, and I. Araújo. User-steering of hpc workflows: State-of-the-art and future directions. In *Proceedings of the 2Nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, pages 4:1–4:6, 2013.

[76] M. Mattoso, C. Werner, G. Travassos, V. Braganholo, E. Ogasawara, D. Oliveira, S. Cruz, W. Martinho, and L. Murta. Towards supporting the life cycle of large scale scientific experiments. In *Int. J. Business Process Integration and Management*, volume 5, pages 79–82. 2010.

[77] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, reloaded. In *Int. Conf. on Scientific and Statistical Database Management*, pages 471–481, 2010.

[78] A. Nagavaram, G. Agrawal, M. A. Freitas, K. H. Telu, G. Mehta, R. G. Mayani, and E. Deelman. A cloud-based dynamic workflow for mass spectrometry data analysis. In *IEEE 7th Int. Conf. on E-Science (e-Science)*, pages 47–54, 2011.

[79] D. Nguyen and N. Thoai. Ebc: Application-level migration on multi-site cloud. In *Int. Conf. on Systems and Informatics (ICSAI)*, pages 876–880, 2012.

[80] B. Nicolae, G. Antoniu, L. Bougé, D. Moise, and A. Carpen-Amarie. Blobseer: Next-generation data management for large scale infrastructures. *Journal of Parallel and Distributed Computing*, 71(2):169–184, 2011.

[81] K. A. Ocaña, D. Oliveira, E. Ogasawara, A. M. Dávila, A. A. Lima, and M. Mattoso. Sciphy: A cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In *Advances in Bioinformatics and Computational Biology*, volume 6832, pages 66–70. 2011.

[82] K. A. C. S. Ocaña, D. Oliveira, F. Horta, J. Dias, E. Ogasawara, and M. Mattoso. Exploring molecular evolution reconstruction using a parallel cloud based scientific workflow. In *Advances in Bioinformatics and Computational Biology*, volume 7409, pages 179–191. 2012.

[83] E. S. Ogasawara, D. de Oliveira, P. Valduriez, J. Dias, F. Porto, and M. Mattoso. An algebraic approach for data-centric scientific workflows. *Proceedings of the VLDB Endowment (PVLDB)*, 4(12):1328–1339, 2011.

[84] E. S. Ogasawara, J. Dias, V. Silva, F. S. Chirigati, D. de Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience*, 25(16):2327–2341, 2013.

[85] T. Oinn, P. Li, D. B. Kell, C. Goble, A. Goderis, M. Greenwood, D. Hull, R. Stevens, D. Turi, and J. Zhao. Taverna/mygrid: Aligning a workflow system with the life sciences community. In *Workflows for e-Science*, pages 300–319. 2007.

[86] T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[87] D. D. Oliveira, K. A. C. S. Ocaña, E. Ogasawara, J. Dias, J. Gonçalves, F. Baião, and M. Mattoso. Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows. *Future Generation Computer Systems*, 29(7):1816–1825, 2013.

[88] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 1099–1110, 2008.

[89] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer. Groudsim: An event-based simulation framework for computational grids and clouds. In *European Conf. on Parallel Processing (Euro-Par) Workshops*, pages 305–313, 2011.

[90] S. Ostermann, R. Prodan, and T. Fahringer. Extending grids with cloud resource management for scientific computing. In *10th IEEE/ACM Int. Conf. on Grid Computing*, pages 42–49, 2009.

[91] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 2011.

[92] E. Pacitti, R. Akbarinia, and M. E. Dick. *P2P Techniques for Decentralized Applications*. Morgan & Claypool Publishers, 2012.

[93] K. Plankensteiner, R. Prodan, M. Janetschek, T. Fahringer, J. Montagnat, D. Rogers, I. Harvey, I. Taylor, Á. Balaskó, and P. Kacsuk. Fine-grain interoperability of scientific workflows in distributed computing infrastructures. *Journal of Grid Computing*, 11(3):429–455, 2013.

[94] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O'Keefe. A 64-bit, shared disk file system for linux. In *16th IEEE Symposium on Mass Storage Systems*, pages 22–41, 1999.

[95] R. Prodan. Online analysis and runtime steering of dynamic workflows in the askalon grid environment. In *7th IEEE Int. Symposium on Cluster Computing and the Grid (CCGRID)*, pages 389–400, 2007.

[96] I. Raicu, Y. Zhao, I. T. Foster, and A. S. Szalay. Data diffusion: Dynamic resource provision and data-aware scheduling for data intensive applications. *The Computing Research Repository (CoRR)*, abs/0808.3535, 2008.

[97] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. Scheduling data-intensiveworkflows onto storage-constrained distributed resources. In *7th IEEE Int. Symposium on Cluster Computing and the Grid (CCGRID)*, pages 401–409, 2007.

[98] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, G. Mehta, F. Silva, and K. Vahi. Online fault and anomaly detection for large-scale scientific workflows. In *13th IEEE Int. Conf. on High Performance Computing and Communications (HPCC)*, pages 373–381, 2011.

[99] R. Sandberg, D. Golgberg, S. Kleiman, D. Walsh, and B. Lyon. Innovations in internetworking. chapter Design and Implementation of the Sun Network Filesystem, pages 379–390. 1988.

[100] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *1st USENIX Conf. on File and Storage Technologies*, 2002.

[101] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta. Workflow task clustering for best effort systems with pegasus. In *15th ACM Mardi Gras Conf.: From Lightweight Mash-ups to Lambda Grids: Understanding the Spectrum of Distributed Computing Requirements, Applications, Tools, Infrastructures, Interoperability, and the Incremental Adoption of Key Capabilities*, pages 9:1–9:8, 2008.

[102] S. Smanchat, M. Indrawan, S. Ling, C. Enticott, and D. Abramson. Scheduling multiple parameter sweep workflow instances on the grid. In *5th IEEE Int. Conf. on e-Science*, pages 300–306, 2009.

[103] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. MIT Press, 1998.

[104] M. Tanaka and O. Tatebe. Workflow scheduling to minimize data movement using multi-constraint graph partitioning. In *12th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (Ccgrid)*, pages 65–72, 2012.

[105] I. Taylor, M. Shields, I. Wang, and A. Harrison. The triana workflow environment: Architecture and applications. In *Workflows for e-Science*, pages 320–339. Springer, 2007.

[106] H. Topcuouglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[107] R. Tudoran, A. Costan, G. Antoniu, and H. Soncu. Tomusblobs: Towards communication-efficient storage for mapreduce applications in azure. In *12th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 427–434, 2012.

[108] P. U-chupala, P. Uthayopas, K. Ichikawa, S. Date, and H. Abe. An implementation of a multi-site virtual cluster cloud. In *10th Int. Joint Conf. on Computer Science and Software Engineering (JCSSE)*, pages 155–159, 2013.

[109] W. M. P. v. d. Aalst, M. Weske, and G. Wirtz. Advanced topics in workflow management: Issues, requirements, and solutions. *Transactions of the SDPS*, 7(3):49–77, 2003.

[110] K. Vahi, I. Harvey, T. Samak, D. Gunter, K. Evans, D. Rogers, I. Taylor, M. Goode, F. Silva, E. Al-Shakarchi, G. Mehta, A. Jones, and E. Deelman. A general approach to real-time workflow monitoring. In *Supercomputing (SC) Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*, pages 108–118, 2012.

[111] J. Wang and I. Altintas. Early cloud experiences with the kepler scientific workflow system. In *Int. Conf. on Computational Science (ICCS)*, volume 9, pages 1630–1634, 2012.

[112] J. Wang, D. Crawl, and I. Altintas. Kepler + hadoop: A general architecture facilitating data-intensive applications in scientific workflow systems. In *4th Workshop on Workflows in Support of Large-Scale Science*, pages 12:1–12:8, 2009.

[113] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2009.

[114] P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour. *Service Level Agreements for Cloud Computing*. Springer, 2011.

[115] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[116] K. Wolstencroft, R. Haines, D. Fellows, A. R. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. P. B. Vargas, S. Sufi, and C. A. Goble. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(Webserver-Issue):557–561, 2013.

[117] J. M. Wozniak, T. G. Armstrong, K. Maheshwari, E. L. Lusk, D. S. Katz, M. Wilde, and I. T. Foster. Turbine: A distributed-memory dataflow engine for extreme-scale many-task applications. In *1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, pages 5:1–5:12, 2012.

[118] U. Yildiz, A. Guabtni, and A. H. H. Ngu. Business versus scientific workflows: A comparative study. In *IEEE Congress on Services, Part I, Services I*, pages 340–343, 2009.

[119] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3:171–200, 2005.

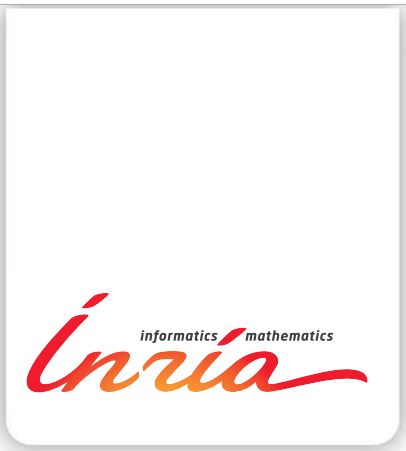
[120] Z. Yu and W. Shi. An adaptive rescheduling strategy for grid workflow applications. In *IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, 2007.

[121] D. Yuan, Y. Yang, X. Liu, and J. Chen. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *IEEE Int. Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, 2010.

[122] H. Zhang, S. Soiland-Reyes, and C. A. Goble. Taverna mobile: Taverna workflows on android. *The Computing Research Repository (CoRR)*, abs/1309.2787, 2013.

[123] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *IEEE Int. Conf. on Services Computing - Workshops (SCW)*, pages 199–206, 2007.

[124] Y. Zhao, I. Raicu, and I. T. Foster. Scientific workflow systems for 21st century, new bottle or new wine? In *IEEE Congress on Services, Part I, Services I*, pages 467–471, 2008.




**RESEARCH CENTRE**
**SOPHIA ANTIPOLIS – MÉDITERRANÉE**

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399