Managing Workflows on top of a Cloud Computing Orchestrator for using heterogeneous environments on e-Science

Abel Carrión[a,], Miguel Caballer[a], Ignacio Blanquer[a], Nelson Kotowski[b], Rodrigo Jardim[b], Alberto Martin Rivera Dávila[b]

[a]*Instituto de Instrumentación para Imagen Molecular (I3M)*
Centro mixto CSIC - Universitat Politècnica de València
Camino de Vera s/n, 46022, Valencia
[b]*Computational and Systems Biology Laboratory.*
Oswaldo Cruz Institute.
Rio de Janeiro. RJ, Brazil, 21040-360

Abstract

Scientific Workflows (SWFs) are widely used to model processes in e-Science. SWFs are executed by means of Workflow Management Systems (WMSs), which orchestrate the workload on top of computing infrastructures. The advent of cloud computing infrastructures has opened the door of using on-demand infrastructures to complement or even replace local infrastructures. However, new issues have arisen, such as the integration of hybrid resources or the compromise between infrastructure reutilization and elasticity. In this article we present an ad-hoc solution for managing workflows exploiting the capabilities of cloud orchestrators to deploy resources on demand according to the workload and to combine heterogeneous cloud providers (such as on-premise clouds and public clouds) and traditional infrastructures (clusters) to minimize costs and response time. The work does not propose yet another WMS, but demonstrates the benefits of the integration of cloud orchestration when running complex workflows. The article shows several configuration experiments from a realistic comparative genomics workflow called Orthosearch, to migrate memory-intensive workload to public infrastructures while keeping other blocks of the experiment running locally. The article computes running time and cost suggesting best practices.

Email address: `abcarcol@i3m.upv.es` (Abel Carrión)

Keywords: Workflow, Workflow Management Systems, Cloud Orchestrator, Multi-platform, e-Science, Cloud Computing, Comparative genomics

1. Introduction

Initially proposed in the business context, workflows were adopted by e-Science to model its applications. The execution of these workflow applications comprises many details. A typical workflow is composed of hundreds of tasks that must be executed in a coordinated way. Moreover, all these tasks must be submitted to specific computing resources and the required inputs must be made available to the application. In data intensive applications, the staging of the input files could require transferring huge amounts of data between resources. In a complex scenario like this one, it is possible to identify several single points of failure: the reception of user inputs, the data transfer between tasks, task executions, hardware crashes, etc. Thus, in these cases it is necessary to carry out actions for resuming the execution, such as retrying the data transfer, rescheduling the task or resetting the resources. Software in charge of dealing with all these aspects are called Workflow Management Systems.
As new computing paradigms emerge and infrastructure evolve, so do the WMSs that support these computing back-ends. Traditionally, Scientific Workflow Applications have been extensively deployed in high performance computing infrastructures, such as powerful clusters and supercomputers. Later, a highly distributed infrastructure, the Grid, appeared as an alternative to traditional approaches. In the last years, a new distributed computing paradigm, Cloud Computing, has appeared as another viable [1] platform for running scientific applications. Some of their main features, such as rapid elasticity, resource pooling, and pay per use, are well suited to the nature of scientific applications that experience a variable demand during its execution. In fact, a typical scenario involves the execution of a scientific workflow whose stages or phases have different computational requirements and therefore, a single infrastructure cannot deal with the whole workflow. In order to avoid outsourcing the whole workflow to external resources for a high cost, it is crucial that WMSs offer multi-platform support where only certain parts of the workflow are migrated to external resources. In order to achieve it, legacy WMSs have been updated to support multiple platforms

for the execution of workflow applications, but they cannot benefit from all the features that the cloud computing provides. This is because most legacy WMSs are derived from grid computing projects and thus are optimized for grids [2]. On the other hand, new generation WMSs normally are focused on fully supporting a small number of cloud computing providers and ignore older computing platforms (i.e Grid, cluster and supercomputers). So, in this work we demonstrate how a multi-platform WMS can be developed on top of a cloud orchestration system for executing SWFs on a heterogeneous computing environment (such as on-premise clouds, clusters and public clouds). It is important to remark that the aim of this work is not to provide yet another WMS, but to show the usefulness of cloud orchestrator systems for running complex workflows. As such, the cloud orchestrator used in this work allows on-demand and automatic infrastructure deployment depending on the workload. The infrastructures are contextualized according to the user's requirements and it is possible to use any Virtual Machine Image (VMI) from any source. Based on this, the paper's contributions are evaluated using, as use case, a realistic comparative genomics workflow called Orthosearch with different configurations. These scenarios suggest best practices for minimizing costs and running times.
The remainder of the paper is structured as follows. Firstly, Section 2 offers an overview of the related work found in the literature. Afterwards, Section 3 explains in detail the design of the multi-platform WMS and the cloud orchestrator system. Then, Section 4 introduces the use case for the experiments, the bioinformatics pipeline called Orthosearch. Section 5 explains the different experiments carried out with our WMS as well as an exhaustive analysis of the results. To sum up, conclusions and future working lines are discussed.

2. Related work

Due to the crucial role that workflow applications play in the scientific community, most current WMSs were developed to enable the execution of these applications in grid computing platforms. When the Cloud Computing became mainstream, legacy WMSs were extended to support it while new WMSs were developed around this new paradigm. Although the aim our work is not to offer another yet WMS but an execution system that can be abstracted from WMSs, related work can only be found in the state-of-the-art WMSs. Moreover, given the impact of the cloud computing paradigm in the WMS landscape, we split them into two categories: pre-cloud era WMSs and post-cloud era WMSs.

The following ones belong to the pre-cloud era:

ASKALON [3] is an application development and computing environment whose aim is to simplify the execution of applications that can benefit from the potential of Grid and Cloud infrastructures. Although [4] shows the execution of a meteorological application in public and private clouds (Eucalyptus and Amazon EC2), there is no proof of a multi-platform execution, where different infrastructures are used simultaneously.
Galaxy [5] is an open, web-based approach that facilitates genomics research. It provides a collaborative environment for performing complex analyses, with automatic provenance tracking, allowing the transparent sharing of computational details, intent and context. Its objective is to offer accessible, reproducible and transparent computational research. A Galaxy instance can utilize compute clusters for running jobs, and can be easily interfaced with portable batch system (PBS) or Sun Grid Engine (SGE) clusters. Galaxy can be also instantiated on cloud computing infrastructures, primarily Amazon Elastic Computing Cloud (EC2). The approach used by Galaxy in the cloud consists on deploying a cloud cluster with a particular Galaxy AMI (Amazon Machine Image) at the beginning of the workflow execution. The drawback of this static virtual cluster is the under usage of the resources when processing complex pipelines with variable resource demands.
Taverna [6] is a WMS with a strong focus on bioinformatics where all computational workflow steps are Web Services. Workflows can be designed and executed on local desktop machines through the workbench or through other clients or web interfaces using the server mode. The server accepts requests from many users to execute remote workflows with support of supercomputers, Grids or cloud environments.
MOTEUR [7] is a workflow engine originally designed to run Taverna [6] workflows in European Grid infrastructures. Its main feature is to enable data, service and workflow parallelism during the execution of the workflow. Although designed to efficiently exploit Grid infrastructures, MOTEUR is an agnostic infrastructure workflow enactor. To the best of our knowledge, there are no examples in the literature that show the behaviour of this engine in a cloud or multi-platform scenario.
Pegasus [8] is a mature WMS that combines features such as portability

across a wide range of infrastructures (clusters, grids and clouds), scalability, data management capabilities, exhaustive monitoring and complex workflow restructuring or transformations. It can be used with popular programming languages among the scientific community (such as Java, Python, Perl) through its APIs (application programming interfaces) and also supports submission via web portals. According to [9], in order to deploy Pegasus workflows in the cloud, users have to configure cloud instances as an HTCondor pool. Similar to the Galaxy case, all the resources needed by the workflow are deployed statically. Moreover, the VM image used for worker instances must contain HTCondor, the Pegasus client tools, and the application, and must be configured to contact the submit node to receive jobs. So, users cannot use a VM image of their choice.
SwinDeW-C [10] (Swinburne Decentralised Workflow for Cloud) is a decentralized (based on peer to peer) WMS derived from its predecessor, SwinDeW-G, a decentralized grid workflow system. Due to its decentralized approach, the system excels at QoS management. Moreover, because it inherits the components of a previous grid project, the workflows can be executed on grid and the cloud. SwinDeW-C has been only tested in SwinCloud, a cloud computing environment built on the computing facilities of the Swinburne University of Technology.
Triana [11] is a workflow environment focused at the Web services level. This Web service orientation enables the execution of mixed-component workflows which interconnect WS-RF services, P2P services, Grid services and Cloud services.
VGrADS [12] is a WMS that provides abstract management of grid and cloud resources. The execution system includes fault tolerance and deadline mechanisms. Because the project is more oriented towards batch-driven workflows than data-intensive workflows, the executions can be configured to use advanced reservation of resources.
WS-PGRADE [13] is a generic distributed computing infrastructure gateway framework that provides a workflow-oriented framework that enables the development, execution and monitoring of scientific workflows where the nodes of these workflows can access several infrastructures including clusters, Grids, desktop Grids, academic and commercial clouds. WS-PGRADE leverages the use of a web service based application called the Distributed Computing Infrastructure Bridge. This web application enables workflow management systems to access transparently several infrastructures using the BES interface. The cloud resources that users can access through the DCI Bridge

must be previously registered by the Bridge's administrator (cloud provider endpoint, VM id, VM size, VM quota). From the end-user's point of view, this fact limits the cloud resources that can be accessed. In our solution, the resources are contextualized following the requirements expressed by the user.

In the post-Cloud era we find the following tools:

The Globus Galaxies platform [14] is a group of components that enable the deployment of SaaS(Software as a Service) scientific gateways. The platform leverages the Galaxy [5] workflow system for the execution of scientific workflows; Globus transfer for transferring large amounts of data; Globus Nexus [15] for identity managements and authentication; and other components such as Swift [16] for parallel execution and HTCondor for scheduling. Although Goblus Galaxies implements elastic scaling by providing on-demand cloud computing resources, it mainly supports Amazon Elastic Cloud Computing (EC2).
SciCumulus [17] is a cloud middleware that acts as intermediary between WMSs and cloud infrastructures, promoting the workflow parallelism following the MTC (Many Tasks Computing) paradigm. It makes transparent the complexity behind the management of cloud computing platforms to the scientists and collects distributed provenance data for reproducibility purposes. Analogous to the Galaxy case, the system deploys static virtual clusters for the workflow executions.

The features that distinguish our solution from the previous ones are:

- Multi-platform support (clusters and any kind of clouds) for using different infrastructures simultaneously. Different stages of the workflow can be executed in different platforms.

- Just-in-time and automatic infrastructure deployment when the workflow requires particular resources. The resorces are provisioned at stage level enabling to adapt the number of deployed resources to the special needs of the workflow.

- High-level infrastructure contextualization according to the user's requirements using ANSIBLE devops tool.

- No pre-packaged images are needed so any VMI from any source can be used.

Table 1: Comparative between available solutions.

	Infrastructures	**Multi-platform**	**Resource provisioning**	**VMI customization**
ASKALON	Grid and Cloud	No	Static	No
Galaxy	Cluster and Cloud	No	Static	No
MOTEUR	Any (Grid oriented)	No	No	No
Pegasus	Cluster, Grid and Cloud	Yes	Static	No
SwinDeW-C	Grid and Cloud	No	Static	No
Taverna	Cluster, Grid and Cloud	Yes	Static	No
Triana	Grid and Cloud	Yes	Static	No
VGrADS	Grid and Cloud	No	Reservation	No
WS-PGRADE	Cluster, Grid and Cloud	Yes	Static	No
Globus Galaxies	Cloud(EC2)	No	Static	cloud-init based
SciCumulus	Cloud	No	Static	No
Our work	**Cluster and Cloud**	**Yes**	**Just-in-time**	**ANSIBLE based**

These features are provided through the use of a state-of-the-art cloud orchestrator system. The solution is released under GPL v3 license and it is available for downloading at github (https://github.com/abel-carrion). Table 1 summarizes and compares the features of all the tools reviewed.
The meaning of each column is the following:

- Infrastructures: List of infrastructure types supported.
- Multi-Platform: If the WMS offers the possibility of using several infrastructures simultaneously in a single workflow execution.
- Resource provisioning: The way in which resources are provided. It can be 'Static' if all the resources needed by the workflow are leased before the beginning of the execution, 'Just in time' if the resources are requested adaptively only when they are actually used, and 'Reservation' of resources if the deployment is batch-oriented instead of data-oriented.
- VMI customization: Specifies the type of customization support provided by the tool.

3. System architecture

The aim of this section is to describe the design and implementation of the architecture behind the WMS developed. The overall organization of the system is depicted in Figure 1. This schema is slightly based on the one showed in [18], one of the most cited papers about the taxonomy of

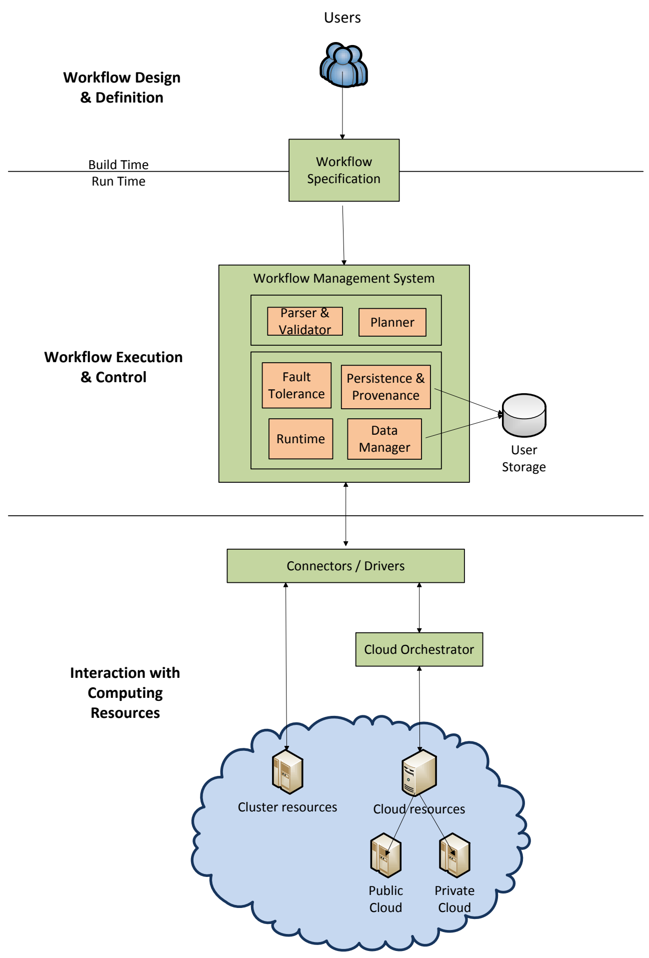

Figure 1: WMS architecture.

Grid WMSs. Our architecture is similar but extended to support an heterogeneous environment. In fact, our WMS currently supports execution on clusters, public clouds (Amazon EC2, Google Cloud Platform and Microsoft Azure), private clouds (OpenNebula and OpenStack) and federated cloud environments (such as EGI FedCloud or FogBow). The next subsections describe each element involved in the architecture.

3.1. Workflow structure

Most scientific applications can be modelled using the workflow programming model. In this model, the application is composed of multiple tasks that

are connected according to their dependencies. In our system workflows are expressed using Directed Acyclic Graphs (DAGs) where the nodes represent computational tasks and the directed edges the dependencies between them. A task has dependencies in the form of files and it will start its execution only when the output file(s) of the task(s) it depends on are available.
There are variations of the model in which the workflow also contains conditional branches (a task may be executed or not depending on the result of previous tasks) and loops (the execution of a part of the workflow is repeated). This workflow structure is known as non-DAG. Although there are WMSs that provide conditional and loop functionalities, the workflow language complexity is higher and therefore its adoption might be limited.

3.2. Workflow composition system

Workflow composition systems are designed to enable users to compose their workflows. For that purpose, systems must provide a high level view of the workflow applications, hiding the complex aspects of the underlying infrastructures. Following the taxonomy of workflow composition systems showed in [18], there are two types: *user-directed* and *automatic*. User-directed composition systems allow users to edit workflows directly while automatic composition systems generate workflows for users automatically. In turn, user directed can be split into two categories: *language-based modelling* and *graph-based modelling*. Our WMS provides user-directed composition where users use language-based modeling.
While almost every state-of-the-art workflow uses markup languages, such as the XML format, for expressing workflows, we have chosen Java Script Object Notation (JSON) [19]. JSON offers some benefits over XML: it is less verbose, easier to write and read for humans and does not require writing end tags.

3.3. Workflow specification

A workflow specification (also called workflow model) defines a workflow including its task definition and structure (task connectivity). There are two types of workflow models: abstract and concrete (executable).

3.3.1. Abstract workflow

The abstract model describes the workflow in an abstract form without referring to the specific computing platforms for task execution. In fact, it is a template that includes the task that must be executed and for each of these tasks, the inputs, outputs, software & hardware requirements, commands and arguments to be used when invoked. To exemplify this, Listing 1 shows a JSON template of a test workflow. The template is interpreted as follows: the workflow contains only one stage named *process0* that must invoked in the resource named *ramses* using as Operating System the *64-bit* version of *Ubuntu*. The desired hardware requirements are 4-single core nodes with 4GB RAM and a 20GB disk. The execution of the stage requires invoking, for each node, the program *test* using as arguments the filenames of *two* files contained in *input0*. The output of the stage (and in this case the output of the workflow) are all the .txt files generated by the program. Notice that all the values that start with '#' are references to JSON objects defined in the same file or the resource configuration file described below.

3.3.2. Resource information file

To transform the abstract workflow into a multi-platform executable workflow, the WMS needs information about the hosts, the environments and the input files of the initial stages of the DAG (i.e. those stages that do not have input dependences with other stages). This information can be found in a JSON configuration file like the one showed in Listing 2 for the previous workflow. The array *hosts* contains a list with the data for accessing the resources, such as: the host name, type, port and different credentials depending on platform to be used (for instance, a certificate for Windows Azure and user/password pair for OpenNebula). Environments is an ad-hoc field for executions on cloud computing platforms. It defines the required features of the VMI to use as a base to create the VMs: Operating System and the software packages that should be installed on it. VMIs are obtained from the image repository associated to each deployment. Last, but not least, the section *inputFiles* declares the input files of the workflow: the identifier, the type (File, Parameter, etc.) and the physical location (URI).

3.4. Workflow conversion

The process of generating an executable workflow based on an abstract workflow is known as workflow conversion. During that process, the original workflow undergoes a series of refinements geared towards optimizing the overall performance and transformations for cloud-support and data management. Thanks to this conversion, the WMS is capable of dynamically leasing and releasing computing resources.

```
{
  "stages": [
    {
      "id": "process0",
      "hostId": "#ramses",
      "environmentId": "#ubuntu64bit",
      "nodes": [
        {
          "numNodes": "4",
          "coresPerNode": "1",
          "memorySize": "4096m",
          "disks": [
            {
              "nDisk": "0",
              "diskSize": "20g"
            }
          ]
        }
      ],
      "execution": [
        {
          "path": "./test",
          "arguments": "#input0(2)"
        }
      ],
      "stageIn": [
        {
          "id": "#input0"
        }
      ],
      "stageOut": [
        {
          "id": "output0",
          "type": "File",
          "filterIn": "*.txt",
          "replica": "none"
        }
      ]
    }
  ]
}
```

Listing 1: Workflow template example

```json
{
  "hosts": [
    {
      "hostId": "ramses",
      "type": "Cloud",
      "subType": "OpenNebula",
      "hostName": "ramses.i3m.upv.es",
      "port": "1111",
      "credentials": {
        "userName": "userName",
        "password": "passWord"
      }
    }
  ],
  "environments": [
    {
      "environmentId": "ubuntu64bit",
      "osName": "linux",
      "arch": "x86_64",
      "osFlavour": "ubuntu",
      "osVersion": "14.04",
      "packages": [
        "unzip"
      ]
    }
  ],
  "inputFiles": [
    {
      "id": "input0",
      "type": "File",
      "values": [
        "db.zip"
      ],
      "extract": "true"
    }
  ]
}
```

Listing 2: Configuration file

As Figure 2 displays, the mapper performs sequential transformations to the abstract workflow.

1. Fusion of two or more sequential tasks (for simplifying the flow), given

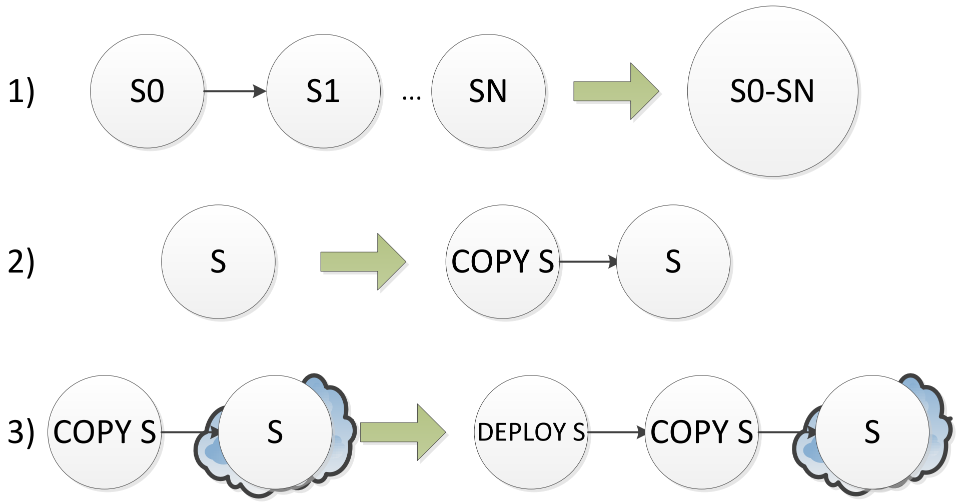

Figure 2: Workflow planner conversions.

the following conditions:

   (a) All the stages are executed on the same infrastructure with the same environment.
   (b) Only the first stage has 0 or more input dependencies with stages that do not belong to the sequence.
   (c) Only the last stage has 0 or more output dependencies with stages that do not belong to the sequence.

2. Addition of COPY and COPYOUT stages, for data management purposes, before every stage obtained in the previous point and after each final stage, respectively.
3. Addition of DEPLOY and UNDEPLOY stages taking into account:
   (a) Stage S is executed in a cloud computing environment
   (b) DEPLOY stages are created before COPY stages.
   (c) UNDEPLOY stages for stage S are added after all the subsequent COPY stages which stage-in products of S have finished.
4. Addition of CLEANUP stages for deleting data after the execution in a cluster.

Figure 3 shows the mapping process of a sample workflow with 5 sequential tasks where stages S0 and S1 are executed on the same cloud platform

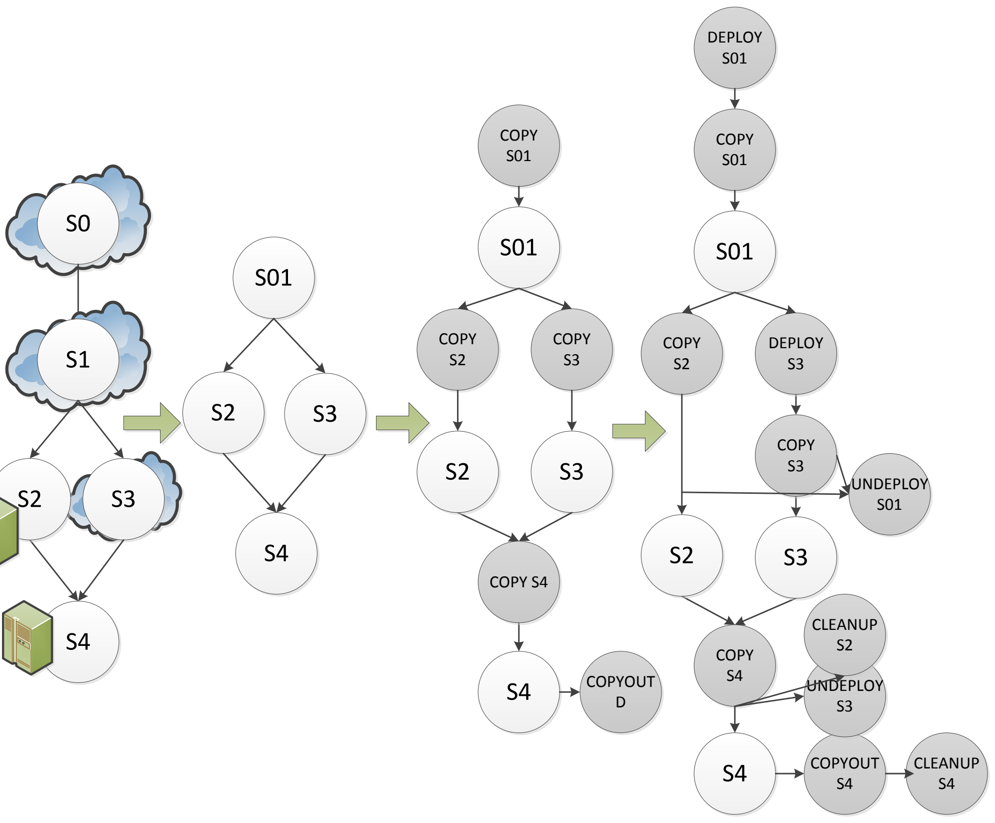

Figure 3: Transformation steps of an abstract workflow into an executable workflow. From left to right: 0(abstract workflow), 1(fusion), 2(data stages), 3(cloud ad-hoc stages)

with the same environment, S2 and S4 are run on a cluster and S3 is executed also in the cloud.

3.5. Workflow execution

Once the mapper has produced the executable, it is submitted to the workflow execution engine. The execution of the workflow begins with the initialization of every element: the state of the tasks are set to *IDLE* and the state of the inputs/outputs to *DISABLED*. Next, due to the data-flow nature of the workflow system, the inputs provided by the user are *ENABLED*, allowing the execution of the first task(s). The workflow execution engine is controlled by two core functions: *runTask* and *getStatus*. The runtime checks if all the inputs of a task are enabled, calling *runTask* in that case. When a task is submitted, the engine periodically monitors its status through the *getStatus* function and if it has finished successfully, enables the outputs of the tasks (which in turn are normally inputs of the next tasks). Obviously, the behaviour of runTask and getStatus will vary according to infrastructure (cluster and cloud) and the task type (deploy, copy, user-defined, undeploy, cleanup or copyout). The next sections explain the functionality of runTask and getStatus for each task type.

3.5.1. Deploy task execution

The execution of a deploy task is required when the user desires to execute a task of the abstract workflow in a cloud platform. In order to dynamically deploy cloud computing resources, the system makes a request to the cloud orchestrator system, the Infrastructure Manager(IM) [20]. The IM is a cloud computing orchestrator that eases the use of IaaS (Infrastructure as a Service) clouds by automating the VMI selection, deployment, configuration, software installation, monitoring and update of Virtual Appliances. The main features of this tool are:

- A language specification of software and hardware requirements for the user applications that can be used by both non-expert (since it is easy to encapsulate recipes as building blocks) and advanced users (due to its high expressivity), called RADL (Resource and Application Description Language) [20].

- Another component, the VMRC (Virtual Machine Resource Catalog) [21] is used to select the most suitable Virtual Machine Image (VMI) based on the user expressed requirements.

- Provision of Virtual Machines on both, public clouds (Amazon EC2, Windows Azure, etc.), private clouds (OpenNebula, OpenStack, etc.) and federated cloud environments (such as EGI FedCloud or FogBow).

- Run-time contextualization of the infrastructure that installs and configures the software required that may not be pre-installed in the VMIs selected, using the Ansible [22] tool.

- Elasticity management support.

- Last but not least, it provides two APIs to enable high-level components to access the functionality: XML-RPC and REST APIs. These APIs provide a set of simple functions for clients to create, destroy, and get

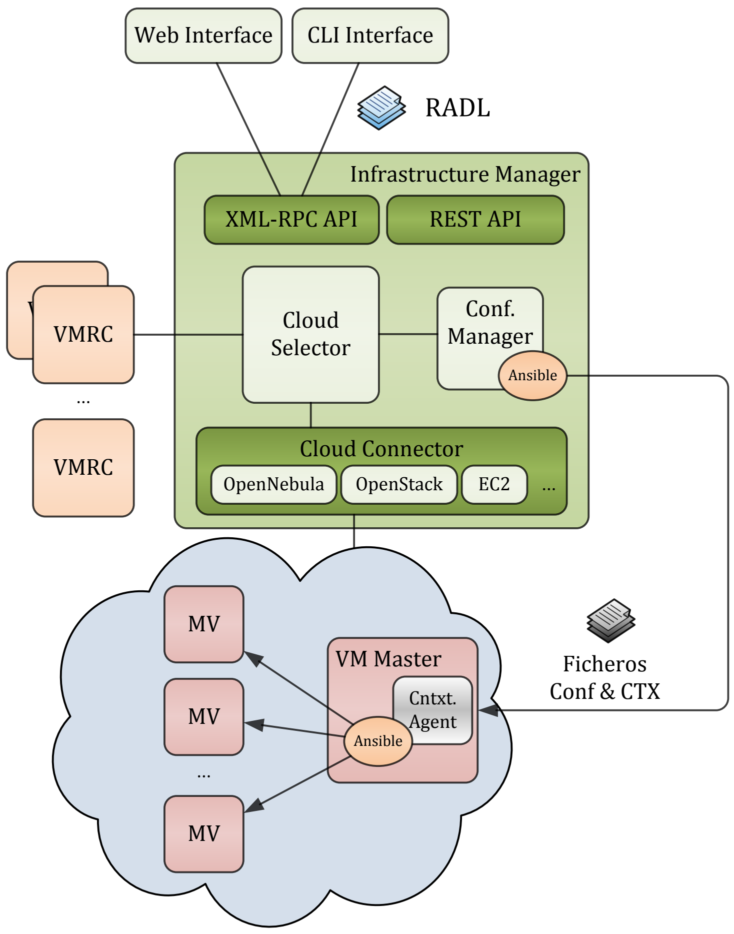

Figure 4: Infrastructure Manager architecture.

information about the infrastructures. The RADL language is used both to create and to get the information about the infrastructures. The IM also provides functions to add and remove resources and modify the features of the existing ones, both hardware and software on run-time.

Figure 4 shows the architecture of the Infrastructure Manager. On the top, the client interfaces currently available for users are depicted (Web and Command Line Interfaces). The IM in the center of the figure provides the upper layers with the functionality through the APIs provided (XML-RPC and REST). The IM uses the "Cloud Selector" component to connect to the VMRC service to get the list of VMIs that best fit the user requirements (expressed in the RADL document) and merge this information with the list of available cloud deployments for the user, in order to get the best option. The "Cloud Connector" layer makes effective the provision of VMs in the cloud deployments. It provides an homogeneous interface to connect with the different cloud middlewares. Finally, once the VMs are deployed and in the running state, the "Configuration Manager" is in charge of managing the contextualization of all the VMs of the infrastructures using the Ansible tool.

In order to request the services of the IM, the WMS uses the API based on the XML-RPC protocol. The runTask function in a deploy task needs to build a RADL document with the hardware and software requirements of the task expressed in the JSON document. Using this RADL document, the WMS invokes the IM to configure the cloud deployment as a Portable Batch System (PBS) cluster where all nodes share the same disk via NFS. In this manner, PBS acts as the scheduler of the jobs that the stage should execute. The *getStatus* invokes the API function that queries the status of the infrastructure. The task is considered to be finished when the status returned by the IM is *configured*. From this point on, the WMS interacts with the cloud infrastructure through SSH, using the information returned by the API call (public IP and user credentials).

3.5.2. Copy task execution

The copy task is in charge of the data management during the execution, one of the most crucial parts of any WMS. These tasks are executed regardless of the computing platform used (cluster or cloud). When *runTask* is called for a copy task, the first step is to declare an unique name for the execution directory (our system uses the current epoch time). Then, this execution identifier is used for creating the execution directory in the file system of the target infrastructure. Now that the execution directory is ready for hosting the task data, the function of *runTask* is staging-in the data. As a convention, our system distinguishes between two types of stage-ins: the ones that begin with the word *input* and the ones that begin with *output*. Inputs are user-provided data while outputs are data whose origin is another task of the workflow (i.e intermediate data).
With respect to the input data, the system can download any file that can

be retrieved with the protocols supported by the unix *wget* command (http, https and ftp). If the URI of the input file defined in the configuration file does not use any of these protocols, the system assumes that the file is in the user local space. Another important issue is the possibility of explicitly indicating that the input files should be extracted on the destination resources. However, since there are tools that require compressed data as input, this extraction should be optional. In any case, the stage-in of an input file triggers the submission of a job to the physical or virtual cluster scheduler for downloading the file and next, if it is required, extracting the file. The system supports almost every popular compression format (.zip, .rar, .gz, .tar). The other type of stage-ins are the intermediate results produced by previous tasks in the DAG. To handle the transference of this kind of data, the WMS submits a basic job that invokes the scp (Secure Copy Protocol) program with the corresponding credentials and arguments.
The goal of *getStatus* in a copy task is to make sure that all the copy jobs submitted by *runTask* have finished successfully. If there is a least one job pending, the status of the copy task returned is *RUNNING*, otherwise the system considers that the task is completed (status FINISHED) and enables the stage-outs of the stage.

3.5.3. User-defined task execution

In contrast to the previous tasks, user-defined tasks are the same that appear in the abstract workflow specification but now they are executable. In our WMS, a user-defined task is said to be executable when two conditions are met: firstly, the target infrastructure is already available (the cluster is accessible or the cloud computing platform is deployed), and secondly, the input data needed by the tasks has been staged-in to these resources. As it can be appreciated, both conditions correspond to the actions performed by the DEPLOY task and COPY task, respectively.
According to the abstract workflow, a task can contain a block of executions or commands to execute. When the *runTask* function is invoked for this kind of tasks, the WMS analyses the commands to determine if there is parallelism in the submission of the job or not. The parallelism of a task is explicitly indicated by the user in the abstract workflow, appending the "*(x)*" expression to an argument where x is the granularity (i.e. the number of files used per job). For instance, let's suppose the scenario showed in Figure 5. The WMS submits a job for each group of two files contained in *db.zip*. If after the analysis the token "*(x)*" is not found, then the system considers that the

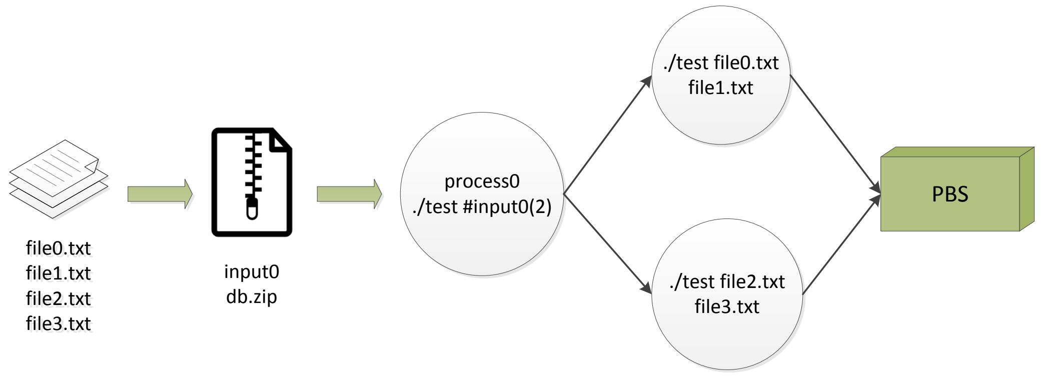

Figure 5: Execution of a parallel task.

task is not parallel and only one job is submitted in that case, passing all the arguments as parameters to the job.

Once *runTask* has submitted all the jobs of the stage to the infrastructure, the goal of *getStatus* is monitoring the status of all jobs until all of them reach a final state (finished or failed).

3.5.4. Undeploy task execution

Due to the variable demand of resources that scientific workflows experience during the execution of the different stages, when a cloud computing task finishes and the output data has been staged-out, the resources assigned to it are no longer needed and they must be freed. Moreover, because of the pay as you go model of this paradigm, the undeployment of resources keeps the user costs down.
As in the deployment task execution case, the *runTask* function calls the proper function of the IM XML-RPC API, *destroyInfrastructure*.
The aim of *getStatus* in this case is to make sure that the infrastructure removal operation is correctly carried out. This is especially important when public clouds are used to avoid incurring in unnecessary costs.

3.5.5. Cleanup task execution

The cleanup task is the equivalent of the undeploy task but for the case of clusters. Because a workflow stage usually generates large amounts of data and clusters are infrastructures shared with other users, a best practice consists on cleaning up the data once it has been staged-out. Thus, the function *runTask* simply deletes via SSH the whole execution directory created for the task and *getStatus* makes sure that the operation is actually done.

3.5.6. Copyout task execution

From the user's point of view, the purpose of the copyout tasks is to retrieve the data products of the computations. The mapper attaches these special tasks only to the final tasks of the abstract workflow specification (i.e tasks which do not have dependencies with other tasks).
The *runTask* function starts the stage-out of the output to one or more locations. The default action is to transfer the data to the user local space (where the submit host is being executed). If besides the field *replica* of the output contains references to another data storage sites, the data will be also copied to these locations. The other function, *getStatus*, will monitor the data transference until all of them are completed.

3.6. Performance optimizations

This section lists a set of optimizations geared towards improving the performance efficiency, in terms of time and costs, of the experiments.

3.6.1. Custom load balancing

When dealing with short running tasks (on the order of minutes or seconds), one of the most common problems of distributed computing infrastructures is the overhead as a consequence of the queueing time on the computing resource schedulers. This fact results on an increase of the response time of the scientific applications. When the WMS executes a parallel stage composed of several tasks, it uses task clustering techniques that group short tasks into coarse-grained tasks, thus greatly reducing the queuing time in the target resources. Our WMS currently implements two clustering techniques, although advanced users can implement and include their own strategies with minimal effort. These are the cluster techniques available by default in the WMS:
Random clustering. This strategy is recommended when the computational cost of processing the input files is similar or unknown. The system computes the clustering granularity, taking into account that: firstly, the number of jobs has to be greater or equal than the number of parallel instances available and secondly, the total estimated execution time of a single job cannot exceed a certain walltime value.
Size clustering. If the runtime of the tasks has a high variance, the previous technique may load balance poorly in some situations, producing clustered

jobs of small tasks and others of larger tasks. In these cases, if the computational load of a task depends on the file size, the size clustering strategy can be used to create jobs with approximately the same total file size (i.e. the same amount of time required to process).

3.6.2. Partial enabling of outputs

If a stage of the workflow executes many trivially parallel jobs, the enabling of the stage-outs can be done in two modes: *standard* and *partial*. The standard mode is the one in which the runtime waits for every parallel job of the stage to have finished successfully, before enabling the stage-outs. On the contrary, in the partial activation mode, the runtime enables a stage-out as soon as a partial output is available. When using cloud computing infrastructures this behaviour can be very effective for overlapping the deployment and copy stages of the next stages while the previous stage is still in execution. Nevertheless, it also increases the usage of the infrastructures.

3.6.3. Prefetching: Partial enabling stages

Similar to the partial enabling of outputs, in some cases, it could be interesting to allow the partial enabling of a stage (i.e. the stage is considered by the runtime as ENABLED when at least one of its input dependencies is ENABLED). As it will be showed below, in the experimentation section, this functionality is useful for pre-fetching input data to the next stages of the workflow. This feature can be enabled using the field "prefetch=True" inside a stage object in the abstract workflow specification.

3.7. Persistence

As it was mentioned before, we assume that the user has access where the WMS is running and it has permanent connection during the workflow execution. Nevertheless, a typical use case involves executing a scientific workflow composed of stages with a significant computational cost (in the order of days or even weeks) and so, demanding a permanent connection to the user machine is not a viable measure. For that reason, the system includes a persistence layer that periodically saves the state of the workflow, allowing users to interrupt the execution and resume it later. The persistence has been implemented using the NoSQL MongoDB [23]. In addition to the features offered by the NoSQL approach (simplicity of design, horizontal scaling, among others) over the traditional relational databases, MongoDB uses JSON-like documents, favouring the straightforward translation between the workflow descriptions and the database documents.

3.8. Fault tolerance

Because failures in distributed computing environments are common, fault tolerance measures are of most importance. It is necessary to provide the proper fault-tolerance mechanisms to handle failures and support the reliable execution in the presence of software or hardware failures. Due to the difference in terms of requirements between the stages that compose a workflow, the WMS defines different fault tolerance policies for each stage. The policies simply define the number of retries in case of software failure or hardware failure. The user indicates such values in the abstract workflow specification, using the object *retries* and its fields *OnWallTimeExceeded*, *OnSoftwareFailure* and *OnHardwareFailure* inside a stage object. If, for some reason, a task exceeds the maximum number of retries for any type of failure, the execution of the workflow is aborted.

3.9. Provenance

Workflow provenance is crucial for users to be able to follow the evolution of their executions and to determine the cause behind a failure. For that purpose, the WMS implements a custom provenance module that registers in a file the events that occur during the execution. This information is not only useful for monitoring the progress of the experiment but also enables its future reproducibility.

4. Use case: Orthosearch

OrthoSearch (Orthologous Gene Searcher) [24] [25] is a genomics comparative workflow. Initially conceived as a Perl-based routine, it is a profile-protein, reciprocal best hits (RBH) based solution for homology inference among species. It comprises several stages and uses distinct bioinformatics tools, such as Mafft [26] and HMMER [27] which confront an orthologous database with an organism multifasta protein data. The abstract workflow is depicted in Figure 6.

Figure 6 displays that the structure of the Orthosearch pipeline is composed of 8 stages: mafft, fasta2stockholm, hmmbuild, hmmsearch, cat, hmmpress, hmmscan and Reciprocal_Best_Hits.

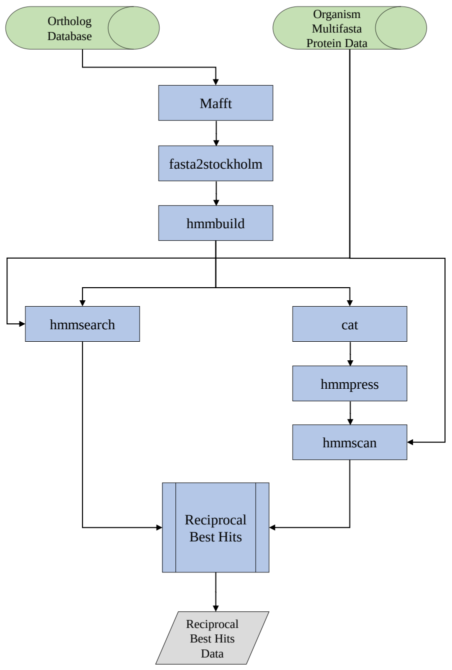

Figure 6: Orthosearch abstract workflow.

5. Experimentation and results

5.1. Data selection

We selected a subset of EggNOG database version 4 [28] which comprises eukaryotic ortholog groups only, EggNOG KOG.
The protozoan specie selected to be confronted with EggNOG KOG database was Cryptosporidium hominis. Cryptosporidium species causes acute gastroenteritis and diarrhea. It is potentially dangerous, with high levels of morbidity and mortality in AIDS patients [29]. In fact, there is no effective treatment or prevention for such infection in humans so far [30].
This protozoan specie is responsible for the death of thousands to millions humans. In addition, there are either no vaccines for such or the available treatments are mostly inadequate due to toxicity and drug resistance [31] [32]. Therefore, comparative genomics experiments among such pathogens genomes that may lead us to a deeper knowledge of these organisms biology are of public health interest. These may aid on the discovery of new issues re-

lated to the pathogenicity of such, as well as help to design new, more specific drugs to treat the infected patients or even prevent the infection itself.

5.2. Infrastructures used

Among the resources that we use for running the experiments, there is a private Cloud that runs OpenNebula and is based on 8 machines, each equiped with 2 processors with 14 core nodes (28 cores per node) and 64 GB of main memory. Therefore the entire infrastructure provides 224 cores and 512 GB of main memory. We also run our experiments on Amazon EC2 using instances of the type m4.xlarge type. Finally, some experiments make use of a cluster named *kahan* with 6 dual processor nodes, where each node contains 2 AMD Opteron processors with 16 cores and 8GB of main memory.

5.3. Sequential execution

The serialized version of the pipeline was entirely executed in two different computing resources with similar performance capabilities: a cluster and a VM instance, both provided with 16 CPU cores, 16GB RAM and 100GB disk.

Figure 7 shows a Gantt chart for the sequential execution of Orthosearch when using the cluster resource while Figure 8 the corresponding chart when using the cloud computing asset.

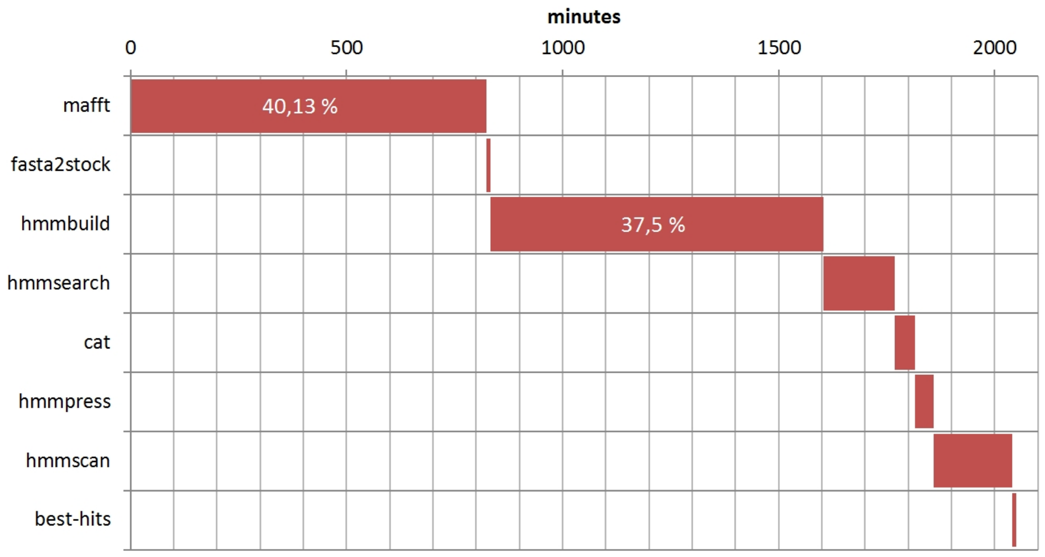

Figure 7: Orthosearch serial execution using a cluster.

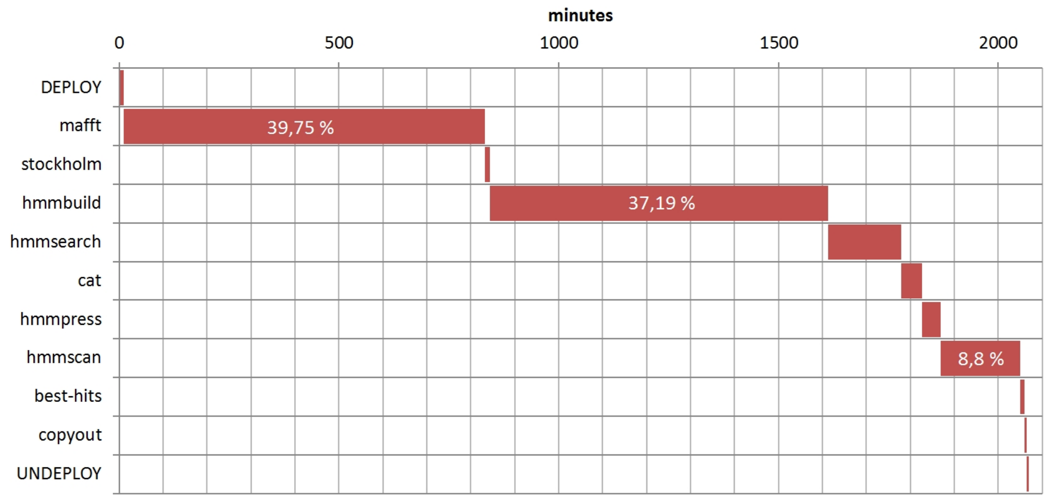

Figure 8: Orthosearch serial execution using a Virtual Machine instance.

From the previous Gantt charts, we extract two interesting facts. Firstly, only three stages of the pipeline take an average of 86,14% of the total time for both scenarios. These computing intensive stages are: mafft, hmmbuild and hmmscan. Secondly, the serial execution of the pipeline in the cloud is slightly slower (0,9%) than the cluster one, as a result of the overheads derived from the deployment and undeployment of the asset and in lesser extent to the use of virtualized resources.

5.4. Cloud Computing WMS-aided execution

The next step of experimentation involved executing the pipeline in a private Open-Nebula based cloud computing infrastructure, using the WMS developed in this work. Table 2 summarizes the configuration defined in the JSON document (abstract workflow) for every stage of the pipeline.

In order to better understand the Gantt charts showed below, Figure 9 depicts the executable workflow generated by the planner component of the WMS after processing the abstract workflow specification. As it can be appreciated, according to the planner optimizations and cloud conversions exposed in previous sections, the 8 original stages of the workflow have been simplified to 5 stages: mafft/fasta2stockholm/hmmbuild; hmmsearch; cat; hmmpress/hmmscan and best-hits. For brevity and clarity, the following Gantt charts cut down the names of the fused stages using only the name of the first stage (i.e hmmpress/hmmscan will be referenced as hmmpress).

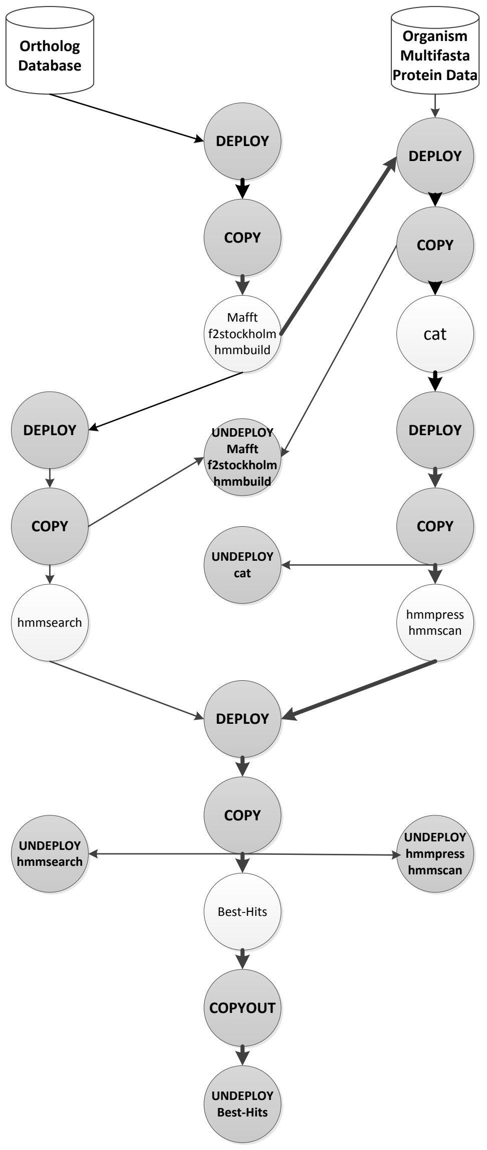

Figure 9: Executable workflow for Orthosearch.

Table 2: Configuration parameters for each Orthosearch stage

	#Node	**Cores/node**	**Memory**	**Disk**	**Parallel**
mafft	16	1	4GB	40GB	Trivially
fasta2stockholm	16	1	4GB	40GB	Trivially
hmmbuild	16	1	4GB	40GB	Trivially
hmmsearch	16	1	4GB	40GB	Trivially
cat	1	1	4GB	40GB	None
hmmpress	1	4	16GB	40GB	None
hmmscan	1	4	16GB	40GB	None
best-hits	1	1	16GB	50GB	None

5.4.1. Execution without pre-fetching

Figure 10 shows the Gantt diagram for the execution of Orthosearch when the pre-fetching option of the WMS is not enabled. In this chart, processing times in the nodes are depicted with red bars while blue bars correspond to data transference actions. The striped pattern in some of the data transference bars (blue) means that it is an intermittent action. As an example, let's examine the "COPY hmmsearch" timeline. The WMS only will copy a hmmsearch input file when a new partial output of mafft is available. After transferring a partial result, the COPY hmmsearch stage will go idle, waiting for a new result from mafft. Finally, the black arrows delimit the time between the deployment and undeployment of a stage and the number of nodes deployed, pointing out the cost associated.

5.4.2. Execution with pre-fetching

The execution of Orthosearch with the pre-fetching option of the WMS enabled can be seen in Figure 11. The main difference with respect the scenario without pre-fetching is that the last stage of the pipeline, best-hits, is activated by the WMS runtime once the first partial result of hmmsearch is available for copying. At the same time, the undeployment of the computing resources associated to hmmsearch is activated sooner. As it is shown below, these differences will have an impact on minimizing the usage of resources.

5.4.3. Performance comparison

The aim of this section is to compare both scenarios (with pre-fetching and without pre-fetching) in terms of makespan or response time of the ex-

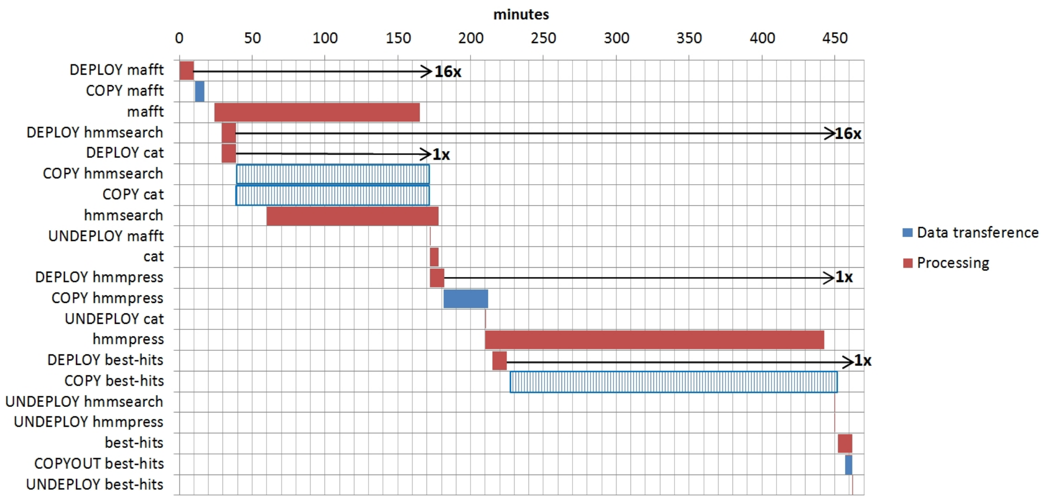

Figure 10: Orthosearch execution using the WMS with pre-fetching not enabled.

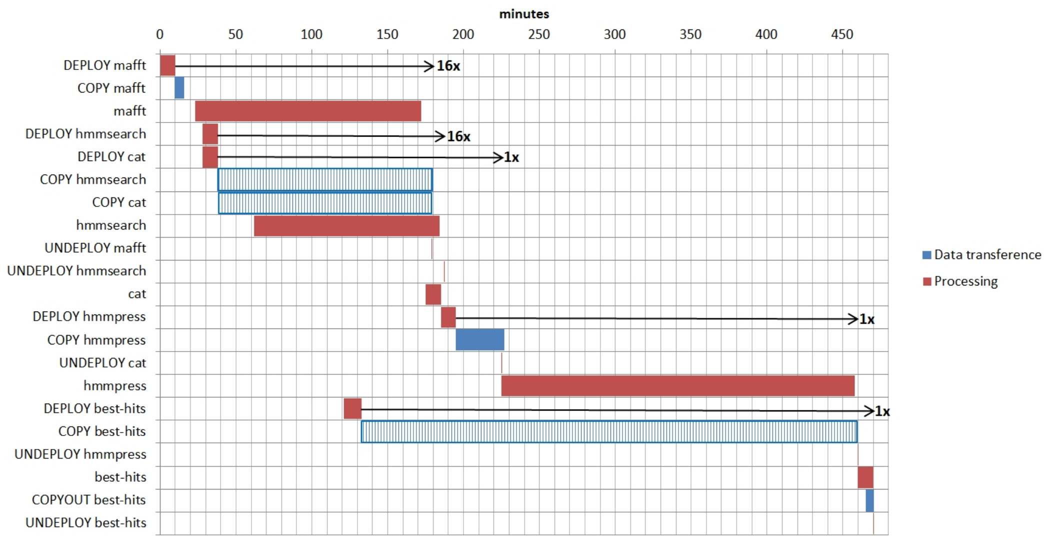

Figure 11: Orthosearch execution using the WMS with pre-fetching enabled.

periment and total CPU usage. With respect to the response time, we see that both scenarios require approximately the same time to be completed: an average of 466.3 minutes. However, when analysing the total CPU usage, we can observe an important difference. The total CPU usage for a stage is the time that the associated resources are deployed (time difference between the end of DEPLOY and the end of UNDEPLOY), linearly weighted with the number of nodes. Graphically, the total CPU is the sum of the longitudes of the black arrows showed in the Gantt chart, individually multiplied by the number of nodes of the stage. According to that, the total CPU usage for the scenario with pre-fetching is 5888 minutes and 9851 for the case without pre-fetching of best-hits (an increase of 67% in resource usage). Thus, to optimize the execution of the experiment, the user has to properly configure the parameters of the WMS, leveraging its empiric knowledge about the behaviour of the workflow being deployed.

5.5. Overall analysis

Figure 12 and Figure 13 show a comparison in terms of makespan (response time) and total CPU usage, respectively, for the 4 scenarios presented in the previous sections: serial execution using a cluster, serial execution in a VM instance and cloud computing WMS-aided executions (with and without pre-fetching). Paying attention to the response time, we can clearly see the benefits of using a distributed approach with the aid of the WMS: a reduction from 2060,7 minutes (about 1 day and 10 hours) to only 466,3 minutes (7,7 hours). Obviously, this speed-up of the makespan comes at the expense of using more computational resources, as it is reflected in the blue bars of the graph. Nevertheless, as it was commented before, it is possible to minimize the resource usage in the WMS case by properly configuring the parameters which control the execution (load balancing, granularity, pre-fetching, etc.).

5.6. Hybrid platform execution

The goal of the present section is to highlight the benefits of using the multi-platform (cluster and cloud) feature of the WMS developed in this work.

5.6.1. Orthosearch's critical path

In order to understand the scenarios presented below, it is necessary to be aware of the critical path for the Orthosearch executable workflow.

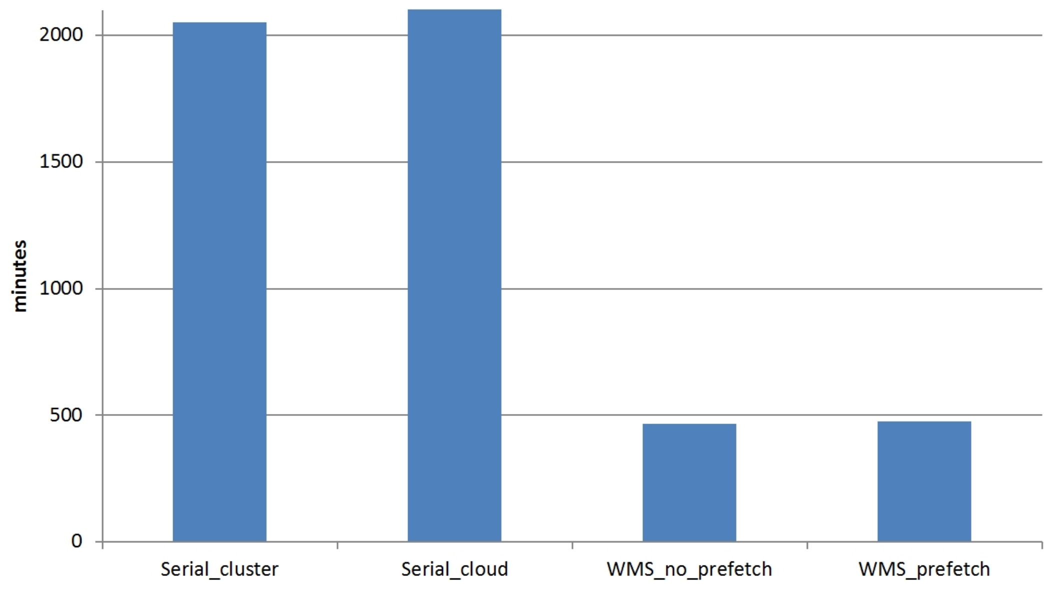

Figure 12: Response time comparison for different scenarios.

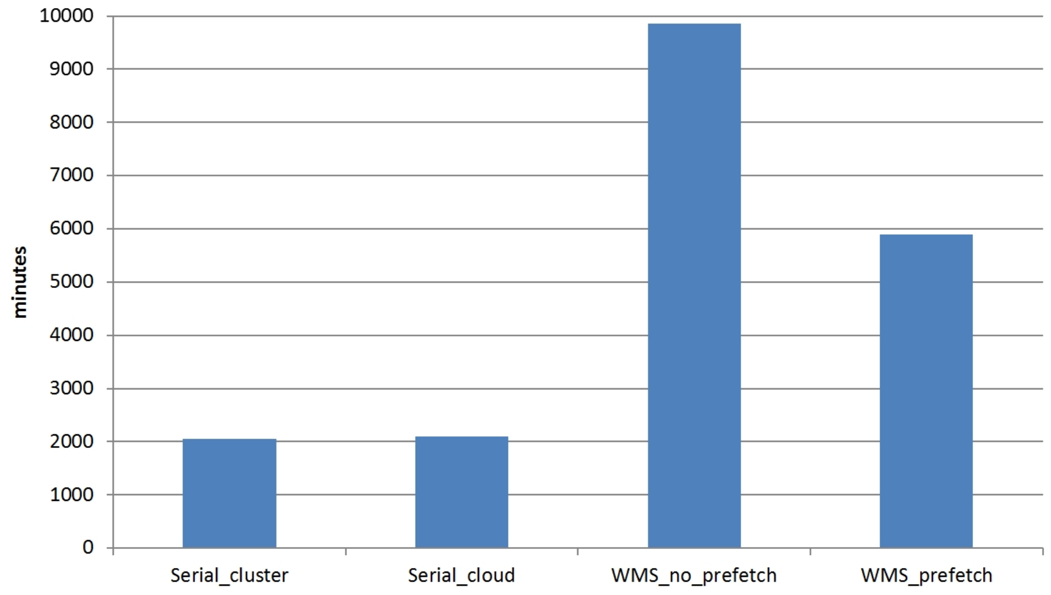

Figure 13: CPU usage comparison for different scenarios.

The critical path can be seen in Figure 9, following the dependency arrows with greater thickness. Thus, the critical path is composed of the following sequence of stages: mafft/fasta2stockholm/hmmbuild - cat - hmmpress/hmmscan - best-hits.

5.6.2. Performance analysis

In Figure 14, there are three time-lines that show the execution time of Orthosearch's critical path for three different scenarios: full cluster on the top, hybrid (cluster and Amazon EC2) in the middle and full cloud (Amazon EC2) in the bottom. In turn, each time-line is divided into three sections or portions: mafft-cat (red bar), hmmpress/hmmscan (green bar) and best-hits (blue bar).
The analysis begins with the execution of Orthosearch in the cluster described above. We can clearly identify that hmmpress/hmmscan is the longest process, taking 69% of the total time. The explanation behind this bottleneck is that the hmmpress and hmmscan processes for the input data selected require about 12GB of memory size to avoid the "swapping" effect and the 'kahan' cluster has only a total of 8GB. Thus, we proceed to optimize the execution by requesting the deployment of this memory-intensive process in a public cloud (Amazon EC2) with a single 16GB memory VM (m4.xlarge type). The results can be checked in the middle bar of Figure 14. Now, because the data to be processed fits in the memory of the EC2 VM (cache effect), the execution time of hmmpress/hmmscan is reduced by 30%, despite of the overheads associated to a cloud execution. Furthermore, we notice that the transference data between different platforms incurs in another overhead that can be mitigated by executing the whole workflow in the public cloud (Amazon EC2), where all the transferences are done between nodes of the same platform, which usually are geographically close. In this way, we achieve a reduction of the total time of 16% with respect the hybrid scenario and 32,43% with respect the full cluster case. Moreover, taking into account that the cluster is an infrastructure with marginal additional operation cost, each reduction of the time achieved in the previous scenarios incurs in a greater cost. Thus, we conclude that full cluster is the slowest scenario but the cheapest one; full cloud is the fastest one but also the most expensive; and the hybrid scenario is a compromise of both.

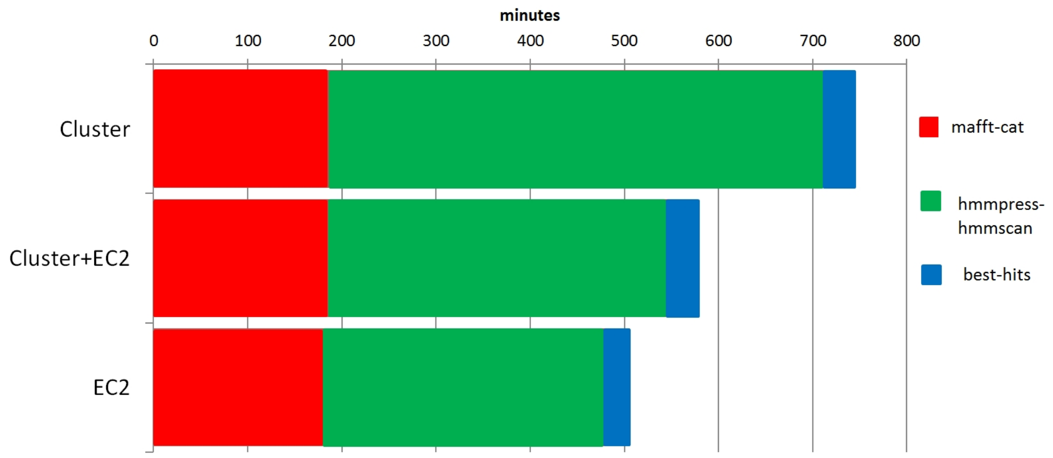

Figure 14: Time-line of Orthosearch's critical path for 3 scenarios.

6. Conclusions and future directions

In the context of e-Science, the effective and efficient use of all the available computational resources is becoming increasingly important for performing scientific research, due to the overflowing amount of data being generated. Because e-Science processes are modelled with workflows, software components called Workflow Management Systems (WMSs) play a crucial role in this data deluge scenario.
Moreover, the advent of Cloud Computing and its core characteristics (rapid elasticity, resource pooling, and pay-per-use, among others) are well-suited to the nature of scientific applications that experience a variable demand during its execution. As a consequence, many WMSs derived from projects in the area of grid computing were updated to support the execution on Cloud resources. However, many of their features are optimized for grids and thus are unable to offer the most key aspects. On the other hand, new generation WMSs normally are focused on fully supporting a small number of cloud computing providers and ignore older computing platforms. In this work, our aim is not to offer yet another WMS but to highlight the usefulness of cloud orchestration systems for developing a multi-platform WMS that adaptively executes SWFs on a heterogeneous computing environment (clusters and different flavours of clouds). In fact, the cloud orchestration system can be ported to any state-of-the-art WMS.
The tool developed in this work has been successfully tested using a comparative genomics pipeline, called Orthosearch. An exhaustive analysis has been

performed, using several scenarios with different configurations. The current working line entails adding support for using Grid computing infrastructures.

Acknowledgments

This paper wants to acknowledge the support of the EUBrazilCC project, funded by the European Commission (STREP 614048) and the Brazilian MCT/CNPq N° 13/2012 , for the use of its infrastructure. The authors would like also to thank the Spanish "Ministerio de Economía y Competitividad" for the project "Clusters Virtuales Elásticos y Migrables sobre Infraestructuras Cloud Híbridas" with reference TIN2013-44390-R.

References

[1] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good, On the Use of Cloud Computing for Scientific Workflows, in: 2008 IEEE Fourth International Conference on eScience, IEEE, ISBN 978-1-4244-3380-3, 640–645, doi:\bibinfo{doi}{10.1109/eScience.2008.167}, URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4736878`, 2008.

[2] R. N.CalHeiros, R. Buyya, Adaptive Execution of Scientific Workflow Application on Clouds, in: O. Terzo, L. Mossucca (Eds.), Cloud Computing with e-Science Applications, CRC Press, NW, 2015.

[3] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, H. L. Truong, ASKALON: A tool set for cluster and Grid computing, Concurrency Computation Practice and Experience 17 (December 2003) (2005) 143–169, ISSN 15320626, doi:\bibinfo{doi}{10.1002/cpe.929}.

[4] G. A. Morar, F. Sch??ller, S. Ostermann, R. Prodan, G. Mayr, Meteorological simulations in the cloud with the ASKALON environment, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7640 LNCS (2013) 68–78, ISSN 03029743, doi:\bibinfo{doi}{10.1007/978-3-642-36949-0_9}.

[5] J. Goecks, A. Nekrutenko, J. Taylor, Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational

research in the life sciences., Genome biology 11 (2010) R86, ISSN 1465-6906, doi:\bibinfo{doi}{10.1186/gb-2010-11-8-r86}.

[6] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, C. Goble, The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud., Nucleic acids research 41 (Web Server issue) (2013) W557–61, ISSN 1362-4962, doi:\bibinfo{doi}{10.1093/nar/gkt328}, URL `http://nar.oxfordjournals.org/content/early/2013/05/02/nar.gkt328.short`.

[7] T. Glatard, J. Montagnat, D. Lingrand, X. Pennec, Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR, International Journal of High Performance Computing Applications 22 (2008) 347–360, ISSN 1094-3420, doi:\bibinfo{doi}{10.1177/1094342008096067}, URL `http://hpc.sagepub.com/cgi/content/abstract/22/3/347`.

[8] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger, Pegasus, a workflow management system for science automation, Future Generation Computer Systems 46 (2015) 17–35, ISSN 0167739X, doi:\bibinfo{doi}{10.1016/j.future.2014.10.008}, URL `http://www.sciencedirect.com/science/article/pii/S0167739X14002015`.

[9] E. Deelman, K. Vahi, M. Rynge, G. Juve, R. Mayani, R. F. da Silva, Pegasus in the Cloud: Science Automation through Workflow Technologies, IEEE Internet Computing 20 (2016) 70–76, ISSN 1089-7801, doi:\bibinfo{doi}{10.1109/MIC.2016.15}, URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7373501`.

[10] X. Liu, D. Yuan, G. Zhang, J. Chen, Y. Yang, SwinDeW-C: a peer-to-peer based cloud workflow system, Handbook of Cloud Computing (2010) 1–24doi:\bibinfo{doi}{10.1007/978-1-4419-6524-0_13}, URL `http://link.springer.com/chapter/10.1007/978-1-4419-6524-0\_13`.

[11] I. Taylor, M. Shields, I. Wang, A. Harrison, The triana workflow environment: Architecture and applications, Workflows for e-Science: Scientific Workflows for Grids (2007) 320–339doi:\bibinfo{doi}{10.1007/978-1-84628-757-2_20}.

[12] L. Ramakrishnan, T. M. Huang, K. Thyagaraja, D. Zagorodnov, C. Koelbel, Y.-S. Kee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, VGrADS, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09 (June 2015) (2009) 1, doi:\bibinfo{doi}{10.1145/1654059.1654107}, URL `http://dl.acm.org/citation.cfm?doid=1654059.1654107`.

[13] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, I. Marton, WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities, Journal of Grid Computing 10 (2012) 601–630, ISSN 15707873, doi:\bibinfo{doi}{10.1007/s10723-012-9240-5}.

[14] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, I. Foster, The Globus Galaxies platform: delivering science gateways as a service, Concurrency and Computation: Practice and Experience (October) (2015) n/a–n/a, ISSN 15320626, doi:\bibinfo{doi}{10.1002/cpe.3486}, URL `http://doi.wiley.com/10.1002/cpe.3486`.

[15] R. Ananthakrishnan, J. Bryan, K. Chard, I. Foster, T. Howe, M. Lidman, S. Tuecke, Globus Nexus: An identity, profile, and group management platform for science gateways and other collaborative science applications, Cluster Computing (CLUSTER), 2013 IEEE International Conference on (2013) 1–3doi:\bibinfo{doi}{10.1109/CLUSTER.2013.6702693}.

[16] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, I. Foster, Swift: A language for distributed parallel scripting, Parallel Computing 37 (2011) 633–652, ISSN 01678191, doi:\bibinfo{doi}{10.1016/j.parco.2011.05.005}, URL `http://dx.doi.org/10.1016/j.parco.2011.05.005`.

[17] D. de Oliveira, E. Ogasawara, F. Baião, M. Mattoso, SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Comput-

ing Paradigm in Scientific Workflows, 2010 IEEE 3rd International Conference on Cloud Computing (2010) 378–385doi:\bibinfo{doi}{10.1109/CLOUD.2010.64}, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5557969.

[18] J. Yu, R. Buyya, A Taxonomy of Workflow Management Systems for Grid Computing, Journal of Grid Computing 3 (3-4) (2006) 171–200, ISSN 1570-7873, doi:\bibinfo{doi}{10.1007/s10723-005-9010-8}, URL http://link.springer.com/10.1007/s10723-005-9010-8.

[19] JavaScript Object Notation, http://json.org/, accessed: 2015-02-24, 2015.

[20] M. Caballer, I. Blanquer, G. Moltó, C. de Alfonso, Dynamic Management of Virtual Infrastructures, Journal of Grid Computing ISSN 1570-7873, doi:\bibinfo{doi}{10.1007/s10723-014-9296-5}, URL http://link.springer.com/10.1007/s10723-014-9296-5.

[21] J. V. Carrión, G. Moltó, C. De Alfonso, M. Caballer, V. Hernández, A Generic Catalog and Repository Service for Virtual Machine Images, 2nd International ICST Conference on Cloud Computing CloudComp 2010 (Vm).

[22] Ansible, http://www.ansible.com/, accessed: 2015-12-21, 2015.

[23] MongoDB, http://www.mongodb.org/, accessed: 2015-02-24, 2015.

[24] S. M. S. da Cruz, M. Mattoso, V. Batista, A. M. R. Dávila, E. Silva, F. Tosta, C. Vilela, M. L. M. Campos, R. Cuadrat, D. Tschoeke, OrthoSearch, in: Proceedings of the 2008 ACM symposium on Applied computing - SAC '08, ACM Press, New York, New York, USA, ISBN 9781595937537, 1282, doi:\bibinfo{doi}{10.1145/1363686.1363983}, URL http://dl.acm.org/citation.cfm?id=1363686.1363983, 2008.

[25] S. M. S. da Cruz, V. Batista, E. Silva, F. Tosta, C. Vilela, R. Cuadrat, D. Tschoeke, A. M. R. Dávila, M. L. M. Campos, M. Mattoso, Detecting distant homologies on protozoans metabolic pathways using scientific workflows., International journal of data mining and bioinformatics 4 (3) (2010) 256–80, ISSN 1748-5673, URL http://www.ncbi.nlm.nih.gov/pubmed/20681479.

[26] K. Katoh, D. M. Standley, MAFFT multiple sequence alignment software version 7: improvements in performance and usability., Molecular biology and evolution 30 (4) (2013) 772–80, ISSN 1537-1719, doi:\bibinfo{doi}{10.1093/molbev/mst010}, URL `http://mbe.oxfordjournals.org/content/30/4/772.short`.

[27] R. D. Finn, J. Clements, S. R. Eddy, HMMER web server: interactive sequence similarity searching., Nucleic acids research 39 (Web Server issue) (2011) W29–37, ISSN 1362-4962, doi:\bibinfo{doi}{10.1093/nar/gkr367}, URL `http://nar.oxfordjournals.org/content/early/2011/05/18/nar.gkr367.short`.

[28] S. Powell, K. Forslund, D. Szklarczyk, K. Trachana, A. Roth, J. Huerta-Cepas, T. Gabaldón, T. Rattei, C. Creevey, M. Kuhn, L. J. Jensen, C. von Mering, P. Bork, eggNOG v4.0: nested orthology inference across 3686 organisms., Nucleic acids research 42 (Database issue) (2014) D231–9, ISSN 1362-4962, doi:\bibinfo{doi}{10.1093/nar/gkt1253}, URL `http://nar.oxfordjournals.org/content/early/2013/11/30/nar.gkt1253.short`.

[29] M. S. Abrahamsen, T. J. Templeton, S. Enomoto, J. E. Abrahante, G. Zhu, C. A. Lancto, M. Deng, C. Liu, G. Widmer, S. Tzipori, G. A. Buck, P. Xu, A. T. Bankier, P. H. Dear, B. A. Konfortov, H. F. Spriggs, L. Iyer, V. Anantharaman, L. Aravind, V. Kapur, Complete genome sequence of the apicomplexan, Cryptosporidium parvum., Science (New York, N.Y.) 304 (5669) (2004) 441–5, ISSN 1095-9203, doi:\bibinfo{doi}{10.1126/science.1094786}, URL `http://www.sciencemag.org/content/304/5669/441.short`.

[30] P. Xu, G. Widmer, Y. Wang, L. S. Ozaki, J. M. Alves, M. G. Serrano, D. Puiu, P. Manque, D. Akiyoshi, A. J. Mackey, W. R. Pearson, P. H. Dear, A. T. Bankier, D. L. Peterson, M. S. Abrahamsen, V. Kapur, S. Tzipori, G. A. Buck, The genome of Cryptosporidium hominis., Nature 431 (7012) (2004) 1107–12, ISSN 1476-4687, doi:\bibinfo{doi}{10.1038/nature02977}, URL `http://dx.doi.org/10.1038/nature02977`.

[31] M. P. Barrett, S. L. Croft, Management of trypanosomiasis and leishmaniasis., British medical bulletin 104 (2012) 175–96,

ISSN 1471-8391, doi:\bibinfo{doi}{10.1093/bmb/lds031}, URL `http://bmb.oxfordjournals.org/content/early/2012/11/22/bmb.lds031.short`.

[32] W. Gatei, C. N. Wamae, C. Mbae, A. Waruru, E. Mulinge, T. Waithera, S. M. Gatika, S. K. Kamwati, G. Revathi, C. A. Hart, Cryptosporidiosis: prevalence, genotype analysis, and symptoms associated with infections in children in Kenya, Am J Trop Med Hyg 75 (1) (2006) 78–82, URL `http://www.ajtmh.org/content/75/1/78.short`.