Instituto de Instrumentación para Imagen Molecular

DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Management of generic and multi-platform workflows for exploiting heterogeneous environments on e-Science

June 2017

Author: Abel Antonio Carrión Collado

Advisors: Prof. Ignacio Blanquer Espert
Dr. Miguel Caballer Fernández

Firstly, I would like to express my sincere gratitude to my advisor Prof. Ignacio Blanquer who provided me an opportunity to join his group. Without his precious support it would not have been possible to conduct this research. His patience, motivation, and knowledge are invaluable.

Besides, I would like to thank my advisor Dr. Miguel Caballer whose guidance helped me in all the time of research and writing of this thesis. In fact, the "heart" of this thesis lies in the results produced by his own thesis.

Also I would like to thank Ignacio Medina for suggesting some ideas that were crucial in the development of this thesis.

I thank my fellow labmates for the stimulating discussions and for all the fun we have had in the last ten years. In particular, I am grateful to Dr. J.Damian Segrelles for his guidance during the final degree project and Prof. Germán Moltó for suggesting some important changes to this document.

Last but not least, I would like to thank my family and friends for supporting me spiritually throughout writing this thesis and my life in general.

Abstract

Scientific Workflows (SWFs) are widely used to model applications in e-Science. In this programming model, scientific applications are described as a set of tasks that have dependencies among them. During the last decades, the execution of scientific workflows has been successfully performed in the available computing infrastructures (supercomputers, clusters and grids) using software programs called Workflow Management Systems (WMSs), which orchestrate the workload on top of these computing infrastructures. However, because each computing infrastructure has its own architecture and each scientific application exploits efficiently one of these infrastructures, it is necessary to organize the way in which they are executed.

WMSs need to get the most out of all the available computing and storage resources. Traditionally, scientific workflow applications have been extensively deployed in high-performance computing infrastructures (such as supercomputers and clusters) and grids. But, in the last years, 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, everything on the basis of cost-efficiency.

The main contribution of this thesis is an ad-hoc solution for managing workflows exploiting the capabilities of cloud computing 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 (supercomputers and clusters) to minimize costs and response time. The thesis does not propose yet another WMS, but demonstrates the benefits of the integration of cloud orchestration when running complex workflows. The thesis shows several configuration experiments and multiple heterogeneous back-ends 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 running time and cost of the experiments is computed and best practices are suggested.

Resumen

Los flujos de trabajo científicos son comúnmente usados para modelar aplicaciones en e-Ciencia. En este modelo de programación, las aplicaciones científicas se describen como un conjunto de tareas que tienen dependencias entre ellas. Durante las últimas décadas, la ejecución de flujos de trabajo científicos se ha llevado a cabo con éxito en las infraestructuras de computación disponibles (supercomputadores, clústers y grids) haciendo uso de programas software llamados Gestores de Flujos de Trabajos, los cuales distribuyen la carga de trabajo en estas infraestructuras de computación. Sin embargo, debido a que cada infraestructura de computación posee su propia arquitectura y cada aplicación científica explota eficientemente una de estas infraestructuras, es necesario organizar la manera en que se ejecutan.

Los Gestores de Flujos de Trabajo necesitan aprovechar al máximo todos los recursos de computación y almacenamiento disponibles. Habitualmente, las aplicaciones científicas de flujos de trabajos han sido ejecutadas en recursos de computación de altas prestaciones (tales como supercomputadores y clústers) y grids. Sin embargo, en los últimos años, la aparición de las infraestructuras de computación en la nube ha posibilitado el uso de infraestructuras bajo demanda para complementar o incluso reemplazar infraestructuras locales. No obstante, este hecho plantea nuevas cuestiones, tales como la integración de recursos híbridos o el compromiso entre la reutilización de la infraestructura y la elasticidad, todo ello teniendo en cuenta que sea eficiente en el coste.

La principal contribución de esta tesis es una solución ad-hoc para gestionar flujos de trabajos explotando las capacidades de los orquestadores de recursos de computación en la nube para desplegar recursos bajo demanda según la carga de trabajo y combinar proveedores de computación en la nube heterogéneos (privados y públicos) e infraestructuras tradicionales (supercomputadores y clústers) para minimizar el coste y el tiempo de respuesta. La tesis no propone otro gestor de flujos de trabajo más, sino que demuestra los beneficios de la integración de la orquestación de la computación en la nube cuando se ejecutan flujos de trabajo complejos. La tesis muestra experimentos con diferentes configuraciones y múltiples plataformas heterogéneas, haciendo uso de un flujo de trabajo real de genómica comparativa llamado Orthosearch, para traspasar cargas de trabajo intensivas de memoria a infraestructuras públicas mientras se mantienen otros bloques del experimento ejecutándose localmente. El tiempo de respuesta y el coste de los experimentos son calculados, además de sugerir buenas prácticas.

Resum

Els fluxos de treball científics són comunament usats per a modelar aplicacions en e-Ciència. En aquest model de programació, les aplicacions científiques es descriuen com un conjunt de tasques que tenen dependències entre elles. Durant les últimes dècades, l'execució de fluxos de treball científics s'ha dut a terme amb èxit en les infraestructures de computació disponibles (supercomputadors, clústers i grids) fent ús de programari anomenat Gestors de Fluxos de Treballs, els quals distribueixen la càrrega de treball en aquestes infraestructures de computació. No obstant açò, a causa que cada infraestructura de computació posseeix la seua pròpia arquitectura i cada aplicació científica explota eficientment una d'aquestes infraestructures, és necessari organitzar la manera en què s'executen.

Els Gestors de Fluxos de Treball necessiten aprofitar al màxim tots els recursos de computació i emmagatzematge disponibles. Habitualment, les aplicacions científiques de fluxos de treballs han sigut executades en recursos de computació d'altes prestacions (tals com supercomputadors i clústers) i grids. No obstant açò, en els últims anys, l'aparició de les infraestructures de computació en el núvol ha possibilitat l'ús d'infraestructures sota demanda per a complementar o fins i tot reemplaçar infraestructures locals. No obstant açò, aquest fet planteja noves qüestions, tals com la integració de recursos híbrids o el compromís entre la reutilització de la infraestructura i l'elasticitat, tot açò tenint en compte que siga eficient en el cost.

La principal contribució d'aquesta tesi és una solució ad-hoc per a gestionar fluxos de treballs explotant les capacitats dels orquestadors de recursos de computació en el núvol per a desplegar recursos baix demanda segons la càrrega de treball i combinar proveïdors de computació en el núvol heterogenis (privats i públics) i infraestructures tradicionals (supercomputadors i clústers) per a minimitzar el cost i el temps de resposta. La tesi no proposa altre gestor de fluxos de treball més, sinó que demostra els beneficis de la integració de l'orquestració de la computació en el núvol quan s'executen fluxos de treball complexos. La tesi mostra experiments amb diferents configuracions i múltiples plataformes heterogènies, fent ús d'un flux de treball real de genòmica comparativa anomenat Orthosearch, per a traspassar càrregues de treball intensives de memòria a infraestructures públiques mentre es mantenen altres blocs de l'experiment executant-se localment. El temps de resposta i el cost dels experiments són calculats, a més de suggerir bones pràctiques.

Contents

Chapter 1

Introduction

Traditional science is representative of two different philosophical trends within the history of science, theoretical (analytical) and experimental (observational). But, in the last decades, Computer Science has revolutionized the way in which science and engineering are conducted and nowadays is recognized as the "third branch" of science along with theory and experimentation [1]. With the inclusion of computing, the term e-Science was defined as "the application of computer technology to the undertaking of modern scientific investigation, including the preparation, experimentation, data collection, results dissemination, and long-term storage and accessibility of all materials generated through the scientific process" [2]. In short, e-Science is the Science in which the use of the computers becomes indispensable for performing scientific research from different scientific areas in an efficient way.

The relation between Science and computing goes back to the 1960s, when powerful computers (in terms of speed calculation and storage capacity), called supercomputers, were employed for performing scientific and engineering problems. At that time, a typical experimental scenario consisted in a repetitive cycle of moving data to a supercomputer for processing, submitting the executions and retrieving the outputs from the data storage [3]. Obviously, it

was necessary to automate this process for allowing scientists to focus on their research and not in the computational management. At the same time the business community was addressing how to automate business processes and as a result the *Workflow* concept was born. In the business context, a Workflow can be defined as the orchestration of a set of activities in order to accomplish a larger and sophisticated goal. A specialization of this idea was adopted by the scientific research to model e-Science processes, the Scientific Workflows (SWFs). In this programming model, scientific applications are described as a set of tasks that have dependencies among them. In this manner, a task will start its execution only when the tasks it depends on have completed their execution.

The execution of workflow applications is a task with many issues. A typical workflow is composed of hundreds of tasks that must be executed in a coordinated way. In addition, 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 demanded by a task could require transferring vast amounts of data among resources. In this complex scenario, it is possible to identify several single points of failure: the reception of user inputs, the data transfer among tasks, tasks executions, hardware crashes, etc. Thus, in all these scenarios it is necessary to carry out actions for resuming the execution, such as retrying the data transfer, rescheduling the task or resetting the resources. The software in charge of dealing with all these aspects are called Workflow Management Systems (WMSs).

As new computing paradigms emerge and infrastructures evolve, so do the WMSs that support these computing back-ends. Scientific workflow applications are deployed in high-performance computing (HPC) infrastructures, such as clusters and supercomputers, and in highly distributed infrastructure, such as the Grid. Grid Computing offers secure and collaborative resource sharing across multiple, geographically distributed institutions. Due to the high impact of Grid infrastructures on the research community, the definition of e-Science was revised as "computationally intensive science that is carried out in highly distributed network environments, or science that uses immense data sets". In the last years, a new distributed computing paradigm, Cloud Computing, has emerged as another viable [4] platform for running scientific applications. Some of its 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, as it may require overcommitting resources on stages where are not needed.

As a consequence of the variable requirements (sequential and parallel execution, data and compute intensive) among the stages of the same workflow that model a scientific application (specially in the bioinformatics field), there is a need for WMSs that efficiently handle the execution of these workflows to enable new research discoveries.

1.1 Motivation

In order to avoid outsourcing the whole workflow to external resources which will lead to highter cost, or if it cannot be performed for IPR (Intellectual Property Rights) or privacy issues, 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 [5]. On the other hand, current WMS supporting clouds are normally focused on fully supporting a small number of cloud computing providers and ignore older computing platforms (i.e Grid, cluster and supercomputers).

So, this thesis shows how a multi-platform WMS can be developed on top of a cloud orchestration system for executing SWFs on a heterogeneous computing environment. The main contributions of this thesis are summarized in the following points:

- An ad-hoc multi-platform WMS developed on top of a cloud orchestration system. It is important to remark that the aim of this thesis is not to provide yet another WMS, but to show the usefulness of cloud orchestrator systems for running complex workflows on a heterogeneous

computing environments (such as on-premise clouds, clusters and public clouds).

- The cloud orchestrator chosen allows on-demand and automatic infrastructure deployment depending on the workflow 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.
- The system is 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.

1.2 Thesis organization

The remaining of the present thesis is structured as follows.

- Chapter 2 presents some basic definitions and terminology related to the topic of the thesis along with a survey of the related state-of-the-art solutions.
- Chapter 3 states the objectives of the thesis and the methodology followed during its development.
- Chapter 4 presents all the aspects regarding the design of the ad-hoc Workflow Management System and how it is binded with the cloud orchestration system.
- Chapter 5 introduces the use case for the experiments, the bioinformatics pipeline called Orthosearch.
- Chapter 6 explains the different experiments carried out with the WMS as well as an exhaustive analysis of the results.
- Chapter 7 contains the conclusions derived from this thesis and future research lines that can be explored in the future.
- Lastly, the final section of the document exposes the main contributions of this thesis regarding to the collaborations established and the literature derived as a result of it.

Chapter 2

State of the Art

This chapter introduces the state of the art in several concepts (from general terms to more specific aspects) related with the topics of the thesis. Section 3.1 begins the chapter with a general description of the different distributed computing paradigms referenced along the text and a comparison between them. Following that, Section 3.2 goes in depth about crucial issues that should be addressed by any system that supports the execution of workflow applications in cloud computing infrastructures. Section 3.3 reviews different orchestration solutions for the management of cloud computing resources and justifies the reason that has led to the choice of one of these systems as a base for the ad-hoc WMS developed in this thesis. Last but not least, Section 3.4 offers a list of the most prominent Workflow Management Systems found in the literature.

2.1 Computing platforms survey

The three main distributed computing paradigms are: cluster, grid and cloud.

2.1.1 Cluster Computing

For many years, high-performance computing (HPC) was restricted to institutions that could afford the significantly expensive supercomputers of that time. But, due to the need of HPC in small scale and at a lower cost, supercomputers were replaced in most cases with clusters [6]. The introduction of cluster platforms was driven by a number of academic projects, such as Beowulf [7], Berkeley NOW [8] and HPVM [9].

> *A cluster is a collection of parallel or distributed computers which are interconnected between themselves through high-speed networks (such as gigabit Ethernet, SCI, Myrinet and Infiniband). They work together in the execution of compute and data intensive tasks that would be not viable to execute on a single computer. Clusters are used for high-availability and load balancing. The high availability is achieved by keeping redundant nodes which are used as backup when components of the system fail. This way, if one node fails there is another idle node which will perform the task, removing single points of failure without any hindrance. When multiple computers are linked together in a cluster, they share computational workload as a single virtual computer. From the users' point of view there are multiple machines, but they function as a single virtual machine [6].*

2.1.2 Grid Computing

The popularity of the Internet and the availability of powerful computers and high-speed network technologies changed the way that computers were used. Grid computing originated in the academia in the mid 1990s with the intention of facilitating users to remotely use idle computing power within other computing centres when the local one was busy. Initially, it only referred to a compute grid and had a rather limited number of users. However, after years of development the grid became mainstream and became an effective way for coordinated resource sharing and problem solving in dynamic, multi-institutional

virtual organizations.

From its inception, Grid computing was conceptually based on the principles of an electric power grid. A large number of electric power generating plants interconnect with one another, providing standardized, reliable, cheap, and ubiquitous access to electric power. Similarly, a computational Grid forms a closed network of a large number of pooled resources providing standardized, reliable, specialized, and pervasive access to high-end computational resources [10]. However, some authors disagree on this analogy with the power grid. Due to the intrinsic heterogeneity of the Grid, its resources offer different characteristics, such as: quality of service, software stack, capability, type of resource, etc. Thus, according to this, the user would be interested not only on the resource itself (like in the power grid scenario) but also on the source from where the resources consumed are provided. Formally, Grid computing [11] combines computers from multiple administrative domains to reach a common goal. One of the main strategies of grid computing is to use middleware to divide parts of a program among several computers. Grid computing involves computation in a distributed fashion, which may also involve the aggregation of large-scale cluster computing-based systems. The size of a grid may vary from a small network of computer workstations within a corporation to large collaborations across many companies and networks.

The definitions given by remarkable people in the field are the following:

> *Buyya et. al. [10] defined grid as a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users quality-of-service requirements.*

> *Ian Foster [12] defined grid as a system that coordinates resources which are not subject to centralized control, using standard, open, general-purpose protocols and interfaces to deliver non-trivial qualities of service.*

2.1.3 Cloud Computing

Cloud Computing [13] is a computing model that emerged around the end of 2007. It provides a pool of computing resources which the users can access through Internet. The basic principle of cloud computing is to shift the computing done from the local computer into the network. Resource are requested on-demand without any prior reservation and thus avoids over-provisioning and improves resource usage.

Currently the most relevant and broadly accepted definition of Cloud Computing is the one provided by the National Institute of Standards and Technology (NIST [14]).

> *According to the NIST, Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

This cloud model implies five essential characteristics, three service models and four deployment models.

NIST defines five characteristics that any deployment must include to be considered a cloud:

- On-demand self-service. Consumers must be able to automatically provision computing resources, with no human interaction required from the provider side.
- Broad network access. Computing resources are accessed through the network by using standard mechanisms, independently of the client platform.
- Resource pooling. Computing resources are pooled together, serving multiple users in a multi-tenant model, reassigning them dynamically based

on the demand. These backend operations are transparent to the user, in the sense that these details are generally concealed to them.

- Rapid elasticity. Resources can be dynamically provisioned and released, in some cases automatically, scaling up and down rapidly. This ability gives the user the illusion of unlimited capacity, adjusting the provision of resources to the system load.

- Measured service. Resource usage can be monitored, controlled and reported at some level of abstraction relevant to the type of resource.

The three basic service models are the following:

- Infrastructure as a Service (IaaS). The provided capability to the user is processing, storage, network and other fundamental computing resources. User has freedom to select the operating system and run arbitrary software on this hardware.

- Platform as a Service (PaaS). The provided capability to the user is a runtime or environment targeted to a particular programming language or applications. User has the freedom to deploy and run applications developed using languages, libraries or tools supported by the provider. Although the user does not have control over the underlying hardware configuration (CPU, memory, etc.) he may have control over configuration settings for the runtime.

- Software as a Service (SaaS). The provided capability to the user is a ready-to-use software service hosted by the cloud platform. These services are accessible over the network using a variety of client devices. Users have no control over the underlying hardware or runtime configuration, although services may provide configurable settings.

Finally, the four deployment models are:

- Public Cloud: The Cloud platform is provisioned for the use of the general public. It may be owned and managed by a single organization, or a combination of them.

- Private Cloud: The Cloud platform is provisioned for the exclusive use by a particular organization. It may be owned and managed by the organization itself, or by third party.

- Community Cloud: The Cloud platform is provisioned for the exclusive use by users of different organizations. It may be owned and managed by one or more of these organizations, by a third party or by any combination of them.
- Hybrid Cloud: The Cloud platform is composed by two or more Cloud deployments, which remain independent from each other and communicate exchanging data and applications using standard or proprietary protocols.

2.1.4 Platform comparison

The purpose of this subsection is to highlight that the ideal platform for executing a scientific workflow application will depend on the software and hardware requirements of each task and the user profile (some have access to supercomputers, others to grids, etc.). Because each platform offers different advantages and disadvantages, there is not an ideal choice for every scenario, and thus, it is crucial to be able to use as many platforms as possible.

When deploying scientific workflow applications on clusters the priority of an execution is to minimize the response time by maximizing the utilization of the resources available for the workflow.

When Grids became widespread, workflows were also deployed on these infrastructures. Due to the highly-distributed nature of Grid resources, the scheduling process became more complex and data movement across wide distances may be necessary. In order to improve the scheduling process, researchers have formulated many efficient scheduling algorithms (mostly based on heuristics). But, even in this case, the focus was on minimizing the execution time of the workflow. Although grids offered a huge amount of resources, their heterogeneity resulted on users being limited to those resources with a software environment capable of supporting their legacy applications. Obviously, Grid providers cannot support the diversity of all possible environments. Moreover, the complexity behind grid infrastructures difficult the design of user-friendly interfaces for scientists without computer science background. In fact, popular middlewares, such as UMD [15] are not easy to use if high-level user interfaces are not provided on top of it.

The advent of Cloud Computing offered another viable platform for running scientific applications [4]. In particular, the use of virtualization provides many useful benefits for scientific applications including: customization of the software stack by the user, performance isolation, check-pointing and migration, better reproducibility of scientific analyses, and enhanced support of legacy applications [5]. Other characteristics of the Cloud such as the elasticity and pay-per-use are well suited to the nature of scientific workflows that experience a variable demand of software and hardware resources during the execution of the different tasks. Because clouds give the perception or illusion of infinite computing resources, the only limitations to the reduction of the execution time are the available resources that the user can afford and the inner scalability of the applications. Therefore, the goal in clouds is to achieve a trade-off between minimizing the execution time and the financial cost.

Security is a feature that becomes more difficult to achieve on new platforms, due to their intrinsic complex model. In this way, clusters are the option recommended for hosting application where sensitive information is managed while clouds are not feasible at all in this particular case. With respect to the costs showed for each platform, we assume that grids are accessed with low (or none) cost granted certificates expended by authorization entities and clouds follow the pay-as-you-go model (minimizing the cost).

2.2 Challenges of the execution of Workflows in Clouds

Every computing paradigm has unique challenges that have the potential to be converted into opportunities for further research. In this section, the challenges of the most recent computing paradigm, cloud computing, are highlighted.

Li et al. [16] identified the following requirements for cloud-enabled workflows:

- *Dynamic resource provisioning*: This is the capability of acquiring and releasing resources as required to allocate the task of workflow.

- *Scalability:* This relates to the capability of reacting to conditions faced during workflow execution to maintain the balance between cost, utilization, and execution time. In the context of this requirement, a change in conditions means adapting to changes in user requirements at runtime. The computing nodes are scaled up and down dynamically by the application.

- *Quality of Service:* Allowing the user to define deadlines is crucial for time-critical workflow applications that need to be completed before a certain amount of time to have value (e.g. applications for prediction of natural disasters, such as floods, cyclones and bushfires). Therefore, the goal is to use the minimum quantity of resources which guarantee that the deadlines are met and costs are not exceeded.

- *Fault tolerance:* This is the possibility to automatically react to changes in the available number of resources or tasks to be processed because of failures. The system developed must be reliable.

- *Security and privacy:* Given that the data being managed by the workflows can be sensitive, mechanisms for protection of the data, either during transfer or once stored in a public cloud, must be available. The applied method should also allow auditing the access and modifications done to the data. Typically, the user has no idea where the data is stored.

- *Multi-tenancy:* When the number of applications running on the same compute node increases, it will reduce the amount of bandwidth allocated to each application which may lead to performance degradation. Fortunately, the VM encapsulation in the cloud infrastructures eases the isolation of the executions of different workflows and users, not existing any interference between them.

- *Provenance:* This requirement involves the capability to collect and process information about the system status and monitor the platform and the application in real time.

As it will be shown later, many of these requirements will be addressed in this thesis.

2.3 Virtual infrastructure deployment and orchestration systems

The aim of this section is to describe a set of virtual management infrastructure systems, a tool that allows the efficient execution of scientific workflow applications in a cloud environment. The list begins from more basic tools that are provided as software layers on top of cloud providers, easing the deployment of virtual infrastructures, to more recent and complex tools that automate the whole life-cycle of an application in the cloud.

Some Cloud providers such as Amazon Web Services (AWS) provide operations for deploying infrastructures. AWS **CloudFormation** [17] gives developers and systems administrators a way to create and manage a collection of related AWS resources, provisioning and updating them. In addition, AWS made available a system called **OpsWorks** [18], an application management service that allows to deploy and operate three-tier (load balancing, logic and database) applications. It allows the contextualization of the VM by specifying: package installation, software configuration and resources such as storage. Both tools emphasize the simplicity of integration with AWS services but at the same time are limited to this cloud provider.

The Nimbus project has developed the **Nimbus Context Broker** [19]. The Context Broker is a service that allows clients to coordinate large virtual cluster launches automatically and repeatably. It is used for deploying what they call "one-click" virtual clusters that function right after launch as opposed to launching a set of "unconnected" virtual machines. It also provides a facility to personalize VMs (seed them with secrets, access policies, and just-in-time configurations).
It is limited to Nimbus clouds and providers that use the Amazon Elastic Cloud Computing (EC2) interface.

Wrangler [20] is a system that automatically provisions and configures virtual clusters in the cloud. The system allows users to send a XML description of the desired virtual cluster to a web service, which manages the provisioning of virtual machines and the deployment of software and services. It is capable of interfacing with many different cloud resource providers (currently it supports

Amazon EC2, Eucalyptus [21], and OpenNebula [22].
Virtual clusters are specified using a custom XML format. The XML format describes virtual clusters as a collection of several nodes, which correspond to virtual machines. Each node has a provider that specifies the cloud resource provider to provision the node from, and defines the characteristics of the virtual machine to be provisioned, such as the VM image to use and the hardware resource type (CPU, memory, disk, etc.). Each node can have multiple roles, which describe the functions that will be performed by the node. Each role is associated with a script, called the role script, that will be executed on the node to configure it for that role. Roles can be customized using parameters, which are passed to the role script when it is executed on the node. Role scripts can be any executable file, but are typically shell, Python or Perl scripts. Users can write their own scripts to implement a custom role.
Although it uses XML as definition language and configuration scripts are provided from outside the VMs, it uses static VM images that require the "wrangler" agent to be pre-installed. Each node can be deployed in a different cloud provider, but the user must indicate the specific details of the provider, such as the instance type, etc.

Vagrant [23] is an automation tool with a domain-specific language (DSL) that is used to automate the creation of VMs and VM environments. The idea is that a user can create a set of instructions, using Vagrant's DSL, that will set up one or more VMs and possibly configure those VMs. Vagrant is composed of the following components. *Providers* are the "back-end" of Vagrant. Vagrant itself does not provide any virtualization functionality; it relies on other products. Providers are how Vagrant interacts with the products that will do the actual virtualization work. A provider could be VirtualBox (included by default with Vagrant), VMware Fusion, Hyper-V, vCloud Air, or AWS. At the heart of Vagrant are *boxes*. Boxes are the predefined images that are used by Vagrant to build the environment according to the instructions provided by the user. A box may be a plain OS installation, or it may be an OS installation plus one or more applications installed. Boxes may support only a single provider or may support multiple providers (for example, a box might only work with VirtualBox, or it might support VirtualBox and VMware Fusion). A single box supports a single provider. The *Vagrantfile* contains the instructions from the user, expressed in Vagrant's DSL, on what the environment should look like, how many VMs, what type of VM, the provider, how they are connected, etc. The Vagrant DSL (and therefore Vagrantfiles) are based on Ruby.

Cloudify [24] is an open source TOSCA-based [25] cloud orchestration software platform written in Python and YAML. Built on a YAML DSL (Domain Specific Language) configuration files called "blueprints" which define the application's configurations, services and their dependencies. With these, Cloudify automates the deployment phases of applications to Cloud computing and Virtualization infrastructure. The blueprints describe how the application interacts with the data center through APIs to execute the defined blueprint configurations.
These blueprint files describe the execution plans for the lifecycle of the application for installing, starting, terminating, orchestrating and monitoring the application stack. Cloudify uses the blueprint as input that describes the deployment plan and is responsible for executing it on the cloud environments. The blueprint also employs cloud driver configuration files as well, to describe machines and their images for the chosen cloud, making it possible to manage the infrastructure as code. For each component it describes the location of the binaries, installation and monitoring configurations. By creating an abstraction layer that isolates the code from the underlying infrastructure, Cloudify is able to support most cloud providers. Cloudify also supports configuration management tools such as Chef [26], Puppet [27] and Ansible [28] for the application deployment phase, as a method of deploying and configuring application services.

Heat [29] implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. A Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans, and can be checked into version control, etc. A native Heat template is being developed, but Heat provides compatibility with the AWS CloudFormation template format, so that many existing CloudFormation templates can be launched on OpenStack [30]. Infrastructure resources that can be described include: servers, volumes, security groups, users, etc. Heat provides both an OpenStack-native REST API and a CloudFormation-compatible Query API. Templates can also specify the relationships between resources (e.g. this volume is connected to this server). This enables Heat to call out to the OpenStack APIs to create the whole infrastructure in the correct order to completely launch the desired application. Although, Heat primarily manages infrastructures, the templates integrate well with software configuration management tools such

as Puppet and Chef.

Cloud Foundry [31] is an open source cloud computing platform as a service (PaaS). When an application is deployed to Cloud Foundry, an image is created for it and stored internally. The image is then deployed in an isolated environment, called Warden container. For multiple instances, multiple images are started on multiple containers. Cloud Foundry's internal Controller uses the BOSH deployment description language to get the underlying infrastructure to spin up virtual machines to run the Warden containers on. When an application is terminated, all its VMs can be recycled for another application to use. If the application instance crashes, its container is killed and a new Warden container is started automatically. A container only ever runs only one application.

The Infrastructure Manager(IM) [32] 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) [32].
- Another component, the VMRC (Virtual Machine Resource Catalog) [33] 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, Microsoft 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 [28] 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 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.

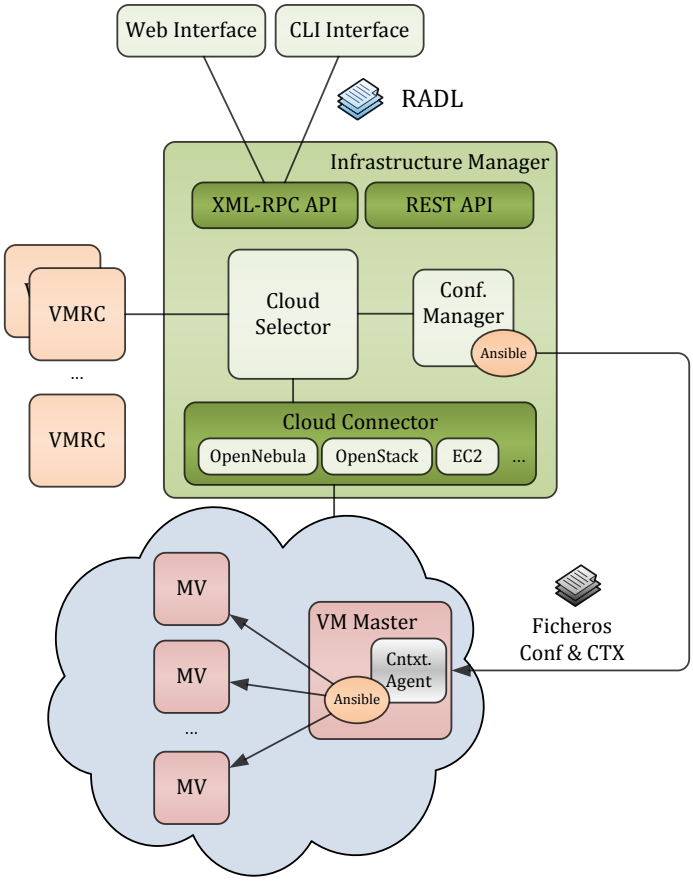

Figure 2.1: Infrastructure Manager architecture.

Figure 2.1 shows the architecture of the Infrastructure Manager. On the top, the client interfaces currently available for users are depicted (Web and Com-

mand 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 utility.

2.4 Related work

Although the aim of the thesis is not to offer yet another 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, they have been split into two categories: pre-cloud era WMSs and post-cloud era WMSs.

The following ones belong to the pre-cloud era:

ASKALON [34] is an application development and computing environment whose initial aim was to simplify the execution of applications that can benefit from the potential of Grid infrastructures. Scientific workflows executed in the ASKALON environment are based on the model described in the AGWL specification language [35]. AGWL documents can express DAGs (Directed Acyclic Graphs) [36] as well as workflow graphs containing loops and conditional branches which impose control. When clouds became mainstream, ASKALON was extended to support executions on cloud computing environments. Although [37] shows the execution of a meteorological application in public and private clouds (Eucalyptus and Amazon EC2), there is no evidence of a multi-platform execution, where different infrastructures are used simultaneously.

Galaxy [38] 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 [39] 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 clusters, supercomputers, Grids or cloud environments. In order to use the different resources, users have to interact with non user-friendly services. Moreover, the execution of the whole workflow can only be deployed in a single infrastructure.

MOTEUR [40] is a workflow engine originally designed to run Taverna [39] 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 [41] 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 [42], 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 [43] (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 simulation environment built on the computing facilities of the Swinburne University of Technology.

Triana [44] is a workflow environment focused at the Web services level. Triana is a workflow environment that consists of a graphical user interface and an underlying subsystem, which allows integration with multiple services and interfaces. Its Web service orientation enables the execution of mixed-component workflows which interconnect WS-RF services, P2P services, Grid services and Cloud services.

VGrADS [45] 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. The virtual grid abstraction of VGrADS unifies workflow execution over batch queue systems (with and without advanced reservations) and cloud computing sites (including Amazon EC2 and Eucalyptus).

WS-PGRADE [46] 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 (DCI) Bridge. This web application enables workflow management systems to access transparently several infrastructures using the Basic Execution Service (BES) [47] 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 [48] is a group of components that enable the deployment of SaaS(Software as a Service) scientific gateways. The platform leverages the Galaxy [38] workflow system for the execution of scientific workflows; Globus transfer for transferring large amounts of data; Globus Nexus for identity managements and authentication; and other components such as Swift [49] for parallel execution and HTCondor for scheduling. Although Globus Galaxies implements elastic scaling by providing on-demand cloud computing resources, this feature works exclusively on the Amazon Elastic Cloud Computing (EC2).

SciCumulus [50] 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.

Table 2.1: Comparison between state-of-the-art WMSs.

	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

Table 2.1 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.

A generalized deficiency of all systems mentioned before is that they not offer just-in-time infrastructure deployment that provisions resources depending on the workflow workload (elasticity through dynamic provisioning). Moreover, almost any system allows resource contextualization according to the user requirements. Therefore, the next chapter exposes the main objectives of this Thesis.

Chapter 3

Objectives and Methods

The aim of this chapter is twofold: to present the objectives of the thesis and the methods followed for the attainment of these goals. Section 3.1 starts exposing the general objective of the thesis, followed by the list of aspects that must be considered, the goals that every aspect must meet and the tasks that must be done to achieve the goals. Next, Section 3.2 details the methods or research plan that has been used as a guide for the development of the tasks of the thesis.

3.1 Objectives

The general objective of the thesis is to demonstrate the benefits of the integration of cloud orchestration in WMSs when running complex workflows. For that purpose, this section proposes the design of an ad-hoc WMS for executing scientific applications on top of a cloud orchestration system. The WMS will exploit key features of the cloud computing paradigm, such as deploying resources on demand, but at the same time it will support the execution of workflows on heterogeneous cloud providers (such as on-premise clouds and public clouds) and traditional infrastructures (supercomputers and clusters) to minimize costs and response time.

Firstly, the ad-hoc WMS must consider all the features regarding the definition of the workflow, and execution management in the desired platforms. Thus, the facets that should be taken into account are:

- **Workflow definition:** Users need a mean to specify the application that they want to execute. For that end, the following data should be provided:
  - **Tasks**: A scientific application presented as a workflow is composed of computation steps or stages that correspond to the tasks of the application.
  - **Task execution order:** The tasks of a scientific application must be executed in a concrete order. In a workflow model this is expressed via dependencies between tasks (i.e a task B has a direct dependence with task A if A must be executed before starting B).
  - **Execution environment**: The user should be allowed to indicate the software configuration required by each task: Operating System (flavour, version, etc.) and the software bundle. Some tasks may have software dependencies different from other tasks, which could be of great importance if we deal with license software. The provisioning cost when dealing with license software in public clouds can be significant and thus, it will be of most importance to minimize the number of compute instances that will use the license software.
  - **Target platform:** Each task of the workflow could be executed in the computing platform that fits better the execution model of the task.
  - **Hardware configuration:** In addition to the platform, it is interesting to be able to specify, for each task, the hardware configuration: the number of nodes, the memory size, disks and capacity, etc.
- **Execution management:** According to the requirements expressed by the user, the system must handle the execution in the corresponding resources with the proper configuration. This facet entails the following actions:

  - **Support of traditional platforms:** Infrastructures (such as supercomputers, clusters and grids) should be supported for executing the workflow tasks.
  - **Cloud Computing support:** The efficient and effective execution of scientific workflows in cloud computing infrastructures requires benefiting from the features provided by cloud orchestrators:
    * **Virtual Machine selection:** According to the software requirements of the user, the rightmost Virtual Machine Image must be chosen.
    * **Infrastructure deployment:** The infrastructure must be set-up and made available for execution.
    * **Virtual Machine contextualization:** Software dependencies of the task must be pre-installed in the virtual machines that will host them.
    * **Just-in-time provisioning and release of resources:** Resources will be provisioned only when they are needed and released when they are no longer necessary .
    * **Use of customized resources:** The resources must reflect the hardware and software requirements indicated by the user.

To achieve these objectives, it will necessary to perform the following tasks:

- To specify a workflow definition language that allows users to describe any scientific application. The language should be easy to understand (as similar as possible to natural language) for non-advanced users but at the same time it should allow to introduce all the relevant data for doing exactly what the user needs to do.
- To design an ad-hoc and multi-platform WMS that allows using traditional infrastructures (supercomputers, clusters and grids) as well as clouds in a efficient and effective way.
- To design a set of drivers for connecting the WMS with the different infrastructures. In particular, all the functionality regarding cloud infrastructures is already provided by the cloud orchestration systems.
- To evaluate the system using a realistic scenario that reflects the benefits achieved with the integration of cloud orchestration in a simple but functional WMS.

3.2 Methods

This section describes the methodology followed during the development of the thesis, in chronological order.

3.2.1 Workflow Management Systems review

The starting point of any research work begins with a survey of the state of the art and the recent development on the field of interest, the Workflow Management Systems. The aim of this survey is to identify functionality gaps in projects that have dealt with similar issues. These projects will help to outline the features of the new system that will "fill" the gaps present in current WMSs. Moreover, the knowledge retrieved from the projects can be used to optimize other features. The expected output of this methodological step is a list of features to be included in the final system (see chapter 2).

3.2.2 Cloud orchestration systems review

Upon reviewing the features of the system to implement, it was detected that all the functionality regarding the efficient and effective use of cloud infrastructures was provided by tools known as cloud orchestration systems. So, in the next step it was imperative to study the state of the art solutions and select the one that fits the requirements of this thesis. After the cloud orchestration system has been chosen, it will tested and studied for future integration with the WMS (see chapter 2).

3.2.3 Workflow specification language methodology

One of the crucial parts of the thesis is to analyse the user requirements for defining the workflows, taking into account that the language specification should be as close as possible to natural language (easing the process to non-advanced users) and versatile. This methodology begins with a revision of the data exchange languages available and the goal is to provide a complete template with the workflow specification in the language chosen (see chapter 4).

3.2.4 Design and implementation of the WMS

The core part of the thesis comprises the design of the ad-hoc and multi-platform WMS. The WMS must account the simplicity of the workflow specification and the support of workflow execution on heterogeneous environments. The outcome of this task is a working prototype of the WMS that implements all the features listed in the first methodological step (see chapter 4).

3.2.5 Experimental testing

The last step consists on the experimental testing of the system in the previous step. A realistic use case will be ideal for demonstrating the capabilities of the system as well as for giving computational support to a concrete scientific problem at the same time (see chapters 5 and 6).

Chapter 4

System architecture

This chapter describes the design and implementation of the architecture behind the ad-hoc WMS developed. The chapter begins with Section 4.1 showing the different parts of the system architecture. Next, Section 4.2 extensively details the workflow specification language how the workflow introduced by the user with the previous language is transformed into something that can be understood and executed by the workflow engine. Next, Section 4.3 exposes various performance optimizations geared towards reducing the turnaround and cost of the experiments. Last but not least, Sections 4.4, 4.5 and 4.6 outline the persistence, fault tolerance and provenance modules of the system, respectively.

4.1 Architecture overview

The system architecture has been designed taking into consideration the objectives set in the previous chapter. The overall organization of the system is presented in [51], where the design is depicted in Figure 4.1. This schema is based on [52], one of the most cited papers about the taxonomy of Grid WMSs. The architecture presented in that paper has been extended in this thesis to support heterogeneous environments. This section begins listing the

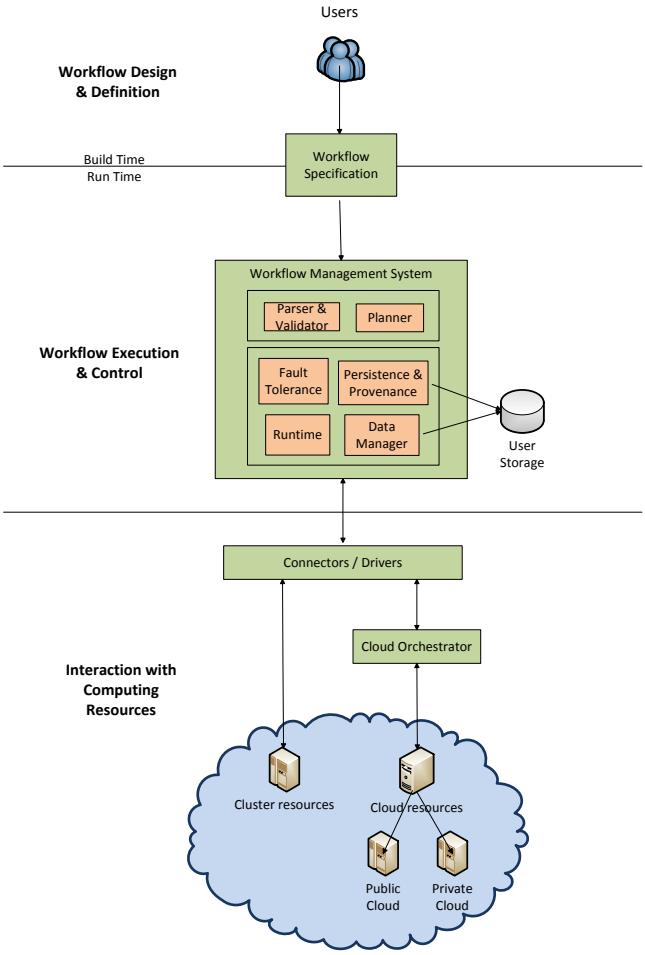

Figure 4.1: WMS architecture overview.

design principles that have guided the definition of the architecture and then describes each one of the components and their role in the management of workflows in heterogeneous environments.

4.1.1 Design principles

The design principles represent a set of guidelines that avoid creating a bad architecture design. If the architecture adheres to the following principles, costs and maintenance will be minimized while usability and extendibility will be promoted. The key principles of the architecture are:

- **Platform-agnostic client**. The client has been developed using a platform-agnostic programming language and thus can be used in major Operating Systems.
- **Generality**. It should be possible to execute any kind of workflow application that can be expressed using the workflow structure explained below.
- **Extensibility**. The architecture can be extended to include new functionality such as support for a new computing and/or storage back-ends.
- **Modularity**. A change on a part of the system should not require changes on the rest of the system if the interfaces are preserved.
- **Multi-platform**. Each stage/node of the workflow can be executed using different computing back-ends.
- **Compliant to the essential characteristics of the NIST Cloud computing definition**. When using cloud resources, the system follows the requirements expressed by the National Institute of Standards and Technology (NIST) cloud computing definition (see Definition 1), with respect to resource provision: on-demand self service, multitenancy and rapid elasticity.

  Definition 1 *"Cloud computing is a model for enabling ubiquitous, convenient,* ***on-demand*** *network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be* ***rapidly provisioned*** *and* ***released*** *with* ***minimal*** *management* ***effort*** *or service provider interaction".*

4.1.2 Components

As Figure 4.1 shows, at the highest level, the functions of Workflow Management Systems could be split into *build time* functions and *run time* functions.

Build time components

The build-time functions comprise the definition and modelling of workflow tasks and their dependencies. Users interact with workflow modelling tools or with the workflow specification directly to generate a workflow specification. This element constitutes the entry point of the system.

Run time components

Run-time functions entail the workflow execution management and the interaction with the computing resources. Initially, the workflow specification generated at built-time is validated by the parser component. If the specification is valid, then the planner component transforms it into an executable workflow that can be used by the runtime element. The main functions of the runtime are: scheduling tasks to jobs, moving the data between resources, restoring the execution flow when a job fails and storing provenance data for reproducibility purposes. The runtime achieves these goals through the following modules: fault tolerance, data manager and persistence and provenance. The interaction of the WMS with the computing and data resources is provided via different connectors or drivers for each back-end.

4.2 Workflow design

The workflow design includes three key factors, namely (a) workflow structure, (b) workflow model/specification and (c) workflow composition system.

4.2.1 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. The workflow structure also known as workflow pattern [53] [54] indicates the temporal relationship between these tasks. Figure 4.2 shows the taxonomy of the workflow structure where the

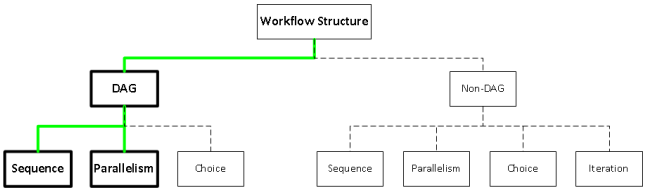

Figure 4.2: Workflow structure taxonomy.

green lines highlight the types supported by the ad-hoc WMS developed in this Thesis. In general, a workflow can be represented as a *DAG* or a *non-DAG*.

The system proposed supports DAG-based workflows (see Figure 4.3. In these workflows, the structure can be classified as *sequence*, *parallelism*, and *choice*. Sequence is defined as an ordered series of computational tasks, with one task starting after a previous task has completed. Parallelism represents tasks which are performed concurrently, rather than serially. In choice control pattern, a task is selected when its associated conditions are true. In the concrete case of the WMS designed, 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. This kind of workflows are called data-driven DAGs.

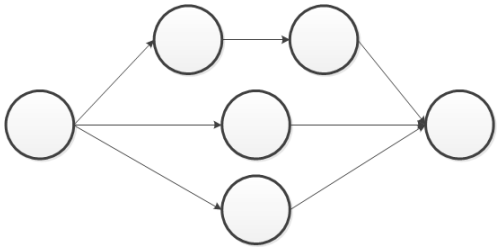

Figure 4.3: A typical scientific application modelled as a DAG.

In addition to all patterns contained in a DAG-based workflow, a non-DAG workflow also includes the *iteration* structure in which sections of workflow tasks in an iteration block are allowed to be repeated. Iteration is also known as *loop* or *cycle*. Although there are WMSs that provide conditional and loop functionalities, the workflow language becomes more complex and therefore its

adoption might be limited.

These four types of workflow structure, namely sequence, parallelism, choice and iteration, can be used to construct many complex workflows.

4.2.2 Workflow Model/Specification

Workflow Model (also called workflow specification) defines a workflow including its task definition and structure. There are two types of workflow models, namely *abstract* workflow and *concrete*. They are also referred to as abstract workflows and concrete workflows [55] [56]. In some literature [57] concrete models are referred to as executable workflows.

In an abstract model, a workflow is described in an abstract form without referring to specific resources for task execution. An abstract model provides a flexible way for users to define workflows without being concerned about low-level implementation details. Tasks in an abstract model are portable and can be mapped onto any suitable platforms at run-time by using suitable discovery and mapping mechanisms. Using abstract models also eases the sharing of workflow description between users working in the same scientific field.

In contrast, a concrete model binds workflow tasks to specific resources. In some cases, a concrete model may include tasks acting as data movement to transfer data in and out of the computation.

Given the dynamic nature of the distributed computing paradigms, it is more suitable for users to define workflow applications in abstract models. However, concrete models may be used by some end users who want to control the execution sequence [58]. A full or partial concrete model can be generated just before or during the workflow execution according to the status of the resources. The proposal for the WMS of this thesis is a model between abstract and concrete, that has been named as *semi-concrete* workflow specification. The model is called semi-concrete because the user must include references to the resources that the task must be mapped to, but at the same time, this workflow is in a non-executable stage (does not contain specific tasks for moving data, deploying resources, etc.).

4.2.3 Workflow Composition System

Workflow composition systems are designed for enabling users to assemble components into workflows. For that purpose, systems must provide a high level view of the workflow applications, hiding the complex aspects of the underlying infrastructures. Figure 4.4 shows the taxonomy for the workflow composition systems. *User-directed* composition systems allow users to edit workflows directly, whereas *automatic* composition systems generate workflows for users automatically. In general, users can use workflow languages for *language-based modelling* and the tools for *graph-based modelling* to compose workflows.

Within language-based modeling, users may express workflow using a markup language such as eXtensible Markup Language (XML) [59] or other formats. Language-based modelling may be convenient for advanced users, but they require to memorize a lot of language-specific syntax. However, workflow languages are more appropriate for sharing and manipulation, whereas the graphical representations are intuitive but they require to be converted into other formats for manipulation. So, workflow languages are designed to bridge the gap between the user interface and the workflow execution engine.

Graph-based modelling allows graphical definition of an arbitrary workflow through a few basic graph elements. It allows users to work with a graphical representation of the workflow. Users can compose and review a workflow by just clicking and dropping the elements of interest. It avoids low-level details and hence enables users to focus on higher levels of abstraction at application level [60]. The major modeling approaches are *Petri Nets* [61], *UML* (Unified Modeling Language) [62] and *user-defined component*. Graph-based modelling is very intuitive and can be handled easily even by a non-expert user. However, the layout of workflow components on a display screen can become very huge and difficult to manage [36].

Another option is to have a system which composes workflows automatically. Compared with user-directed systems, automatic composition systems are ideal for large scale workflows which are very time consuming to compose manually. However, the automatic composition of application components is challenging because it is difficult to capture the functionality of components and data types used by the system.

As Figure 4.4 depicts, the WMS of the thesis provides user-directed composition through language-based modelling. Although XML-based languages are

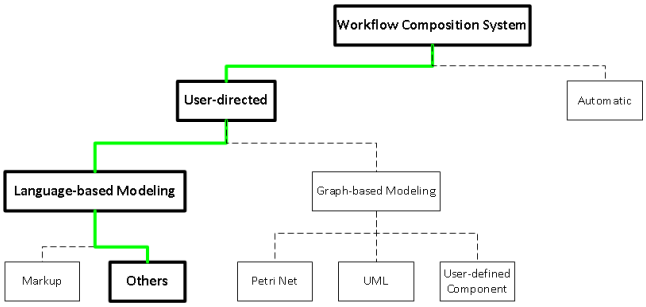

Figure 4.4: Workflow composition system taxonomy.

widely used for workflow specification and many tools are provided to validate the syntax and semantics of an XML document, in this thesis it has been chosen Java Script Object Notation (JSON) [63] as modelling language. JSON is not classified as a markup language and it offers some benefits over XML: it is less verbose, easier to write and read for humans and does not require writing end tags.

4.2.4 Workflow Specification Language

One of the objectives of the Thesis was to allow users to describe the workflow application to execute and the hardware, software and configuration requirements of the resources where the workflow is going to be deployed. For that end, this section describes the design of the Workflow Specification Language for providing the following features:

- It should allow to define the workflow tasks and their dependencies.
- It should address hardware requirements (number of processors, memory, etc), software requirements (bundles, libraries) and configuration parameters of the target resources. This is specially important for shaping the Virtual Machines in Cloud Computing infrastructures.
- It is recommended to use a language and terminology close to the one used by the underlying orchestration system.
- The language should be accessible to non-advanced users.

It is important to remark that workflow tasks in cloud environments entail a set of features that can be split into two categories:

- Features related to the Virtual Machine Image. These features will be used by the cloud orchestration system for searching the optimal image in the repository of Virtual Machine images. Examples of these characteristics are: the Operating System or the applications installed.
- Features related with the deployment of Virtual Machines. For example, memory size or number of CPUs, cores per CPU, etc. These features will be used by the cloud orchestration system for properly deploying the Virtual Machines.

Abstract workflow skeleton

The structure of the abstract workflow is composed of two elements: the resource information file and the semi-abstract workflow instance.

Resource information file

Firstly, to transform the workflow instance into a concrete or executable workflow, the WMS needs the information showed in Listing 1.

```
{
  "comment": "Definition of 1 or more hosts"
  "hosts": [

  ],
  "comment": "Definition of 0 or more software environments"
  "environments": [

  ],
  "comment": "User-provided input files"
  "inputFiles": [

  ]
}
```

Listing 1: Resource information file skeleton.

Therefore, a resource information file contains three sections:

- Information about the front-end hosts of the platforms. The key word "hosts" is used for providing all the information needed to access the different computing platforms through their front-end nodes.
- Execution environments used by the workflow tasks deployed on cloud computing platforms. It defines the required features of the VMI (Virtual Machine Image) to use as a base to create the VMs, such as the Operating System and the software packages that should be installed on it. The cloud orchestration system obtains the VMIs from the image repository associated to each deployment.
- Last but not least, section named with the key word "inputFiles" declares the user-provided input files of the workflow from their local host (where the WMS client is running) or from a remote location.

Below are listed the considered properties for defining an element of each section of the resource information file.

Host properties The host properties that can be defined in the resource information file are represented by the following tokens or key words:

- **hostId:** (string) Acts as a primary key or identifier of the host element in the workflow specification environment. As it will shown below, this identifier will be used for referencing the host object in the semi-abstract workflow.
- **type:**. (string) Nominal property that indicates the infrastructure type. Values allowed are: "Cluster" and "Cloud".
- **subType:**. (string) Value that takes a different meaning depending on the infrastructure type. If type is "Cluster" then "subType" is the type of scheduler (e.g. LRMS, etc.). In the "Cloud" case it refers to the particular cloud provider to use (e.g. OpenNebula, OpenStack, Microsoft Azure, etc.)
- **hostName:** (string) In contrast to "hostId", "hostName" is the canonical name of the host that will be used for connection issues.
- **port:** (4-digit integer) In addition to the hostName it is necessary to know the connection port.

- **credentials:** It indicates how to access the host, either using a pair user/password (like in the OpenNebula case) or a public key. In turn, credentials is composed of the following attributes:
  - **type:** (string) The value can be *user* (pair user/password) or *publicKey*.
  - **userName:** (string) Name of the user.
  - **passWord:** (string) Password access.
  - **publicKey:** (string) Public key for accessing the node.
  - **privateKey:** (string) Private key for accessing the node.

Environment properties The environment features that can be defined in the resource information file are the following:

- **environmentId:** (string) It is the identifier of the whole environment object in the workflow specification context. In the semi-abstract workflow it will be used for referencing the environment information contained in the object.
- **osName:** (string) Operating System Name (e.g. "linux", "windows").
- **arch:** (string) Architecture type. The valid values are: *i686* and *x86_64*.
- **osFlavour:** (string) Operating System flavour (e.g. if osName is "linux" osFlavour can be set to "ubuntu", "debian", "centOS", etc. and if osName is "windows" then osFlavour can be "windows xp", "windows 7", etc.).
- **osVersion:** (string) It must be a string composed of integers separated by dots. For instance: "10.04", "7.1.2".
- **packages:** (string) A list of names of software bundles that must be pre-installed in the Virtual Machines in order to successfully run the applications.

inputFiles properties The input file properties are the ones listed next:

- **id:** (string) Unique identifier of the stageOut element.
- **type:** (string) In the current version, due to the data flow nature of the workflow, the type will set to "File"
- **URI:** (string) Uniform Resource Identifier of the file. It can be a file in the local filesystem or in a remote file server.

Semi-concrete workflow instance

The second part of the abstract workflow is the semi-concrete workflow instance that includes the definition of the tasks and the connectivity between them. Listing 2 shows its general structure.

```
{
  "comment": "Declaration of 1 or more stages",
  "stages": [
    {
      "id": " ",
      "comment": "The mapping task-resource is provided by the user",
      "hostId": " ",
      "environmentId": " ",
      "comment": "Information about the execution nodes",
      "nodes": [

      ],
      "comment": "0 or more command-line executions",
      "execution": [

      ],
      "comment": "0 or more input files of the stage",
      "stageIn": [

      "comment": "0 or more output files of the stage",
      ],
      "stageOut": [

      ]
    }
  ]
}
```

Listing 2: Semi-concrete workflow structure.

In the semi-concrete workflow case, the description comprises one main section: the list of "stages" or workflow tasks. Each stage object contains a set of properties described below and four lists: "nodes", "execution", "stageIn" and "stageOut".

Stage properties The properties that can be defined for a stage element in the semi-abstract workflow are:

- **id:** (string) It is the identifier of the task.
- **hostId:** (string) Reference to the host object, defined in the resource configuration file, that will be in charge of hosting the stage.
- **environmentId:** (string) (optional) Reference to the software environment (defined in the resource information file) which will be used for shaping the execution nodes.
- **nodes:** (list) A list with the information about the nodes that will be used for executing the stage. See below the properties of a node object.
- **execution:** (list) A task can be composed by a group of command-line executions that will be called sequentially. See below the properties of an execution object.
- **stageIn** (list) The list of input files required for the task. See below the properties of a stageIn object
- **stageOut** (list) The list of products or outputs of the stage. See below the properties of a stageOut object.

Node properties The characteristics of a node element are the following:

- **numNodes:** (positive integer) Number of CPUs requested by the task.
- **coresPerNode:** (positive integer) Number of computation cores of each CPU.
- **memorySize:** (positive integer, followed by a character that denotes the unit: *B*-byte, *K*-kilobyte, *M*-megabyte, *G*-gigabyte, *T*-terabyte) The quantity of RAM memory of each node.

- **disks:** An ordered array of disks to reflect the order in the Virtual Machine (disk.0, disk.1, etc.) The disk 0 is a special case because is the boot disk of the system. The attributes considered for a disk are:
  - **nDisk:** (positive integer) The identifier of the disk.
  - **diskSize:** (positive integer, followed by a character that denotes the unit: *B*-byte, *K*-kilobyte, *M*-megabyte, *G*-gigabyte, *T*-terabyte) The amount of memory for the disk with identifier "nDisk".

Execution properties An execution element contains the following properties:

- **path:** (string) Absolute location or relative to the working directory (the one in which the systems logs will be available when accessing the platform) of the executable.
- **arguments:** (string) List of arguments that will be used with the path for invoking the command-line execution. Some arguments can appear as references to stageIn or stageOut elements using the token "#".

stageIn property (semi-abstract workflow) In contrast to the resource information file, the stageIn elements in the semi-abstract workflow only contain a property called "id" which is a reference to either an input file defined in the resource information file or an output file of the present document (intermediate result).

stageOut properties The properties of the stageOut are very similar to the ones defined for the stageIn elements in the resource information file.

- **id:** (string) Unique identifier of the stageOut element.
- **type:** (string) In the current version, due to the data flow nature of the workflow, the type will set to "File".
- **filterIn:** (string) The output files of the stageOut object will be those that match the regular expression.
- **replica:** (string) URI location where the result will be saved for provenance or fault-tolerance purposes.

A workflow specification example

Let's suppose that a user wants to specify the workflow showed in Figure 4.5, composed of 2 stages: A and B. As the Figure shows, the stage A is deployed in a cloud environment while the stage B is executed in a cluster.

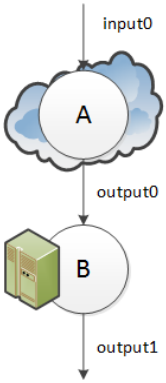

Figure 4.5: Workflow diagram example.

In first place, the user provides the resource information file showed in Listing 3.

```
{
  "hosts": [
    {
      "hostId": "ramses",
      "type": "Cloud",
      "subType": "OpenNebula",
      "hostName": "ramses.i3m.upv.es",
      "port": "1111",
      "credentials": {
        "userName": "username1",
        "passWord": "password2"
      }
    },
    {
      "hostId": "kahan",
      "type": "Cluster",
      "subType": "PBS",
      "hostName": "kahan.dsic.upv.es",
      "port": "9999",
      "credentials": {
        "userName": "userName2",
        "passWord": "passWord2"
      }
    }
  ],
  "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 3: Resource information file example.

The resource information file shows that are two kinds of hosts identified with the labels “ramses” and “kahan”. The host “ramses” is the front-end to an OpenNebula cloud infrastructure while “kahan” is a cluster that provides a PBS (Portable Batch Scheduler) scheduler. Moreover, the user has defined an environment for the cloud platform, called “ubuntu64bit”. According to the information provided in this section, the environment requires installing a 64-bit ubuntu linux Operating System (in concrete, the version 14.04) with the package “unzip” for extracting compressed files. Finally, the inputFiles section indicates that the workflow has only one external dependency labelled as “input0” which contains the “db.zip” file. The property “extract” is used in the context of zipped files for automatically extracting the files upon arrival to their destinations (the local disk of the execution node).

The next step is to define the semi-abstract workflow instance like the one showed in Listing 4.

```
{
  "stages": [
    {
      "id": "A",
      "hostId": "#ramses",
      "environmentId": "#ubuntu64bit",
      "nodes": [
        {
          "numNodes": "4",
          "coresPerNode": "1",
          "memorySize": "4096m",
          "disks": [
            {
              "nDisk": "0",
              "diskSize": "20g"
            }
          ]
        }
      ],
      "execution": [
        {
          "path": "cat",
          "arguments": "#input0(1) >> A.out"
        }
      ],
      "stageIn": [
        {
          "id": "#input0"
        }
      ],
```

```
      "stageOut": [
        {
          "id": "output0",
          "type": "File",
          "filterIn": "A.out",
          "replica": "none"
        }
      ]
    },
    {
      "id": "B",
      "hostId": "kahan",
      "execution": [
        {
          "path": "head",
          "arguments": "-n 10 #output0 > B.out"
        }
      ],
      "stageIn": [
        {
          "id": "#output0"
        }
      ],
      "stageOut": [
        {
          "id": "output1",
          "type": "File",
          "filterIn": "B.out",
          "replica": "none"
        }
      ]
    }
  ]
}
```

Listing 4: Semi-abstract workflow example

In the semi-concrete workflow instance the user has defined two stage elements corresponding to the tasks A and B of the proposed workflow.
On the one hand, the stage A is going to be executed using the host ramses (the OpenNebula cloud front-end defined in the resource configuration file) with the environment "ubuntu64bit". For this first task, 4 single-core nodes with 4GB of memory and a shared disk of 20GB are going to be used. The execution of the task A invokes the unix-command "cat" in each node associated to the task for each single file contained in input0. Then, the first 10 lines of each file are appended to a file named "A.out". The input file of the process is the stageIn input0 defined in the resource configuration file which consisted of a zipped file with filename "db.zip". Obviously, the output of this task is the file "A.out".

On the other hand, the stage B is going to be run in the cluster labelled as "kahan". The limit to the number of nodes in this case is given by the number of physical nodes of the cluster. For that reason, this information should be only provided in the command and not as a field of the specification. The execution of B consists on extracting the first ten lines of the file produced by the task A (A.out) and writing the result in the file "B.out". This last file will be the output of the task B and the workflow.

Workflow parsing and validation
Upon providing the abstract workflow specification and the resource configuration file to the system, both files are examined by a parser to check if the syntax of these documents is compliant to the JSON specification. Because the system is mostly implemented in the Java programming language, we use Jackson [64] for parsing workflow documents.
If the validation passes, the WMS performs a semantic validation of the JSON documents. Among other rules, the semantic validator checks that every reference (to a host, environment or input file) in the abstract workflow exists in the resource configuration file. If the semantic validator finds any error, it prompts to the user the erroneous file and line.
Moreover, the identifiers of the different elements should be unique (two different stages cannot share the same id) and the values should match certain regular expressions (for instance, memory is an integer followed by the characters 'm'(mega), 'g'(giga) or 't'(tera)). Lastly, because the system only supports workflows that can be modelled as DAGs, the module performs an structural validation of the graph to make sure that it does not contain any cycle.

4.3 Workflow planning

Workflow mapping refers to the process of translating abstract workflows to concrete workflows. As shown in Figure 4.6, mapping strategies of workflow applications can be categorized into either *static* or *dynamic*. In a static planning, concrete models have to be generated before the execution according to current information about the execution environment and the dynamically changing (like in the Grid case) is not taken into account. In contrast, a dynamic planning uses both dynamic information and static information about the resources to make scheduling decisions at run-time.

Static planning, also known as full-ahead planning, include user-directed and simulation-based scheduling. In user-directed scheduling, users emulate the scheduling process and make resources mapping decisions according to

their knowledge, preference and/or performance criteria. In simulation-based scheduling, the "best" schedule is achieved by simulating task execution on a given set of resources before a workflow starts execution. The simulation can be processed based on static information or the result of performance estimation.

Dynamic schemes include *prediction-based* and *just in-time* scheduling. Prediction-based dynamic scheduling uses dynamic information in conjunction with some results based on prediction. It is similar to simulation-based static scheduling, in which the scheduler is required to predict the performance of task execution on resources and generate a near optimal schedule for the task before it starts execution. However, it changes the initial schedule dynamically during the execution. Rather than making a schedule ahead, just in-time schedule [56] only makes scheduling decisions at the time of task execution. Dynamic schemes are critical when using Grid resources due to its dynamic nature where utilization and availability of resources changes over time and the optimal resource can join at any time

As it was explained in the Workflow Specification Language section, the WMS developed in this Thesis uses abstract workflows called semi-abstract workflows because the user has to indicate the resources that they want to use for each task of the workflow. This is possible due to the static nature of the environments supported (supercomputers, clusters and clouds). Thus, it uses an static user-directed planning strategy.

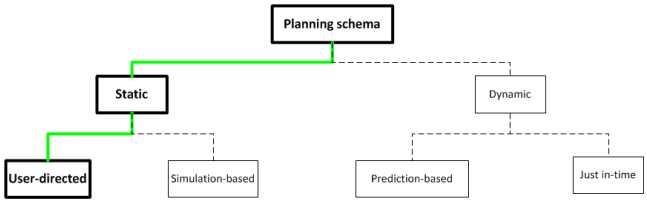

Figure 4.6: Planning schema taxonomy.

4.3.1 Workflow restructuring at build time

In addition to the information about the environment, the mapping (or planning) process of the user's abstract workflow to the final executable workflow requires optimizing the workflow. During this build time process, the underlying initial DAG undergoes a series of refinements geared towards optimizing the overall performance. In additions, transformations are performed for actual cloud computing support and data management. In fact, the workflow restructuring process explained below distinguishes the WMS developed in this Thesis from other systems with similar purposes by providing a novel approach that dynamically provisions and releases cloud computing assets. The following sections detail the process step-by-step.

Stage merging As Figure 4.7 displays, the first refinement fuses two or more sequential tasks if the following conditions are given:

- All the stages are executed on the same infrastructure with the same environment features.
- Only the first stage has input dependencies with tasks different from the ones to be merged.
- Only the last stage has output dependences with other external tasks

Of course, this conversion is not mandatory and its purpose is to optimize or simplify the execution flow.

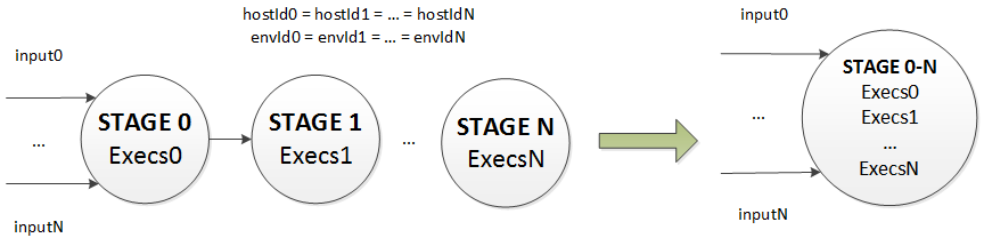

Figure 4.7: Fusion of sequential stages.

Data management tasks The second transformation (see Figure 4.8) adds data management tasks (labelled as COPY) before every task of the workflow obtained in the previous step. The goal of these tasks is to stage-in/out the required input by the tasks or outputs to user selected locations, respectively.

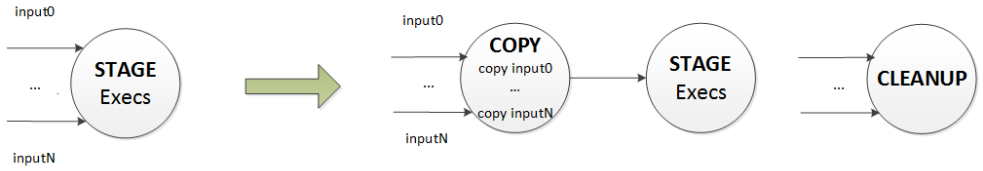

Figure 4.8: Addition of data management tasks.

Cloud computing oriented tasks Finally, if a task is executed on cloud computing resources (see Figure 4.9), it is necessary to add tasks that deploy the resources only when they are needed and undeploys them when the products have been copied to their destinations. In order to accomplish this, the planner adds a synthetic task (called DEPLOY) before the stage-in task produced in the previous step and another task (UNDEPLOY) after the stage-in of subsequent tasks.

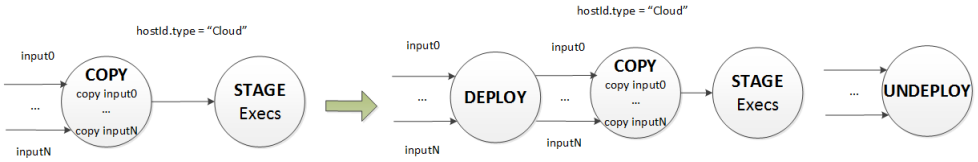

Figure 4.9: Task for the dynamic provisioning and release of resources.

A workflow restructuring example Figure 4.10 shows the restructuring process of a simple workflow with 5 tasks where stages S0 and S1 are executed on the same cloud platform with the same environment specifications, S2 and S4 are executed on a cluster, and S3 is also deployed in a cloud infrastructure. Although the next section will exhaustively explain the implementation of each task, the planner generates five types of of tasks: deploy, copy, undeploy, cleanup and copyout.

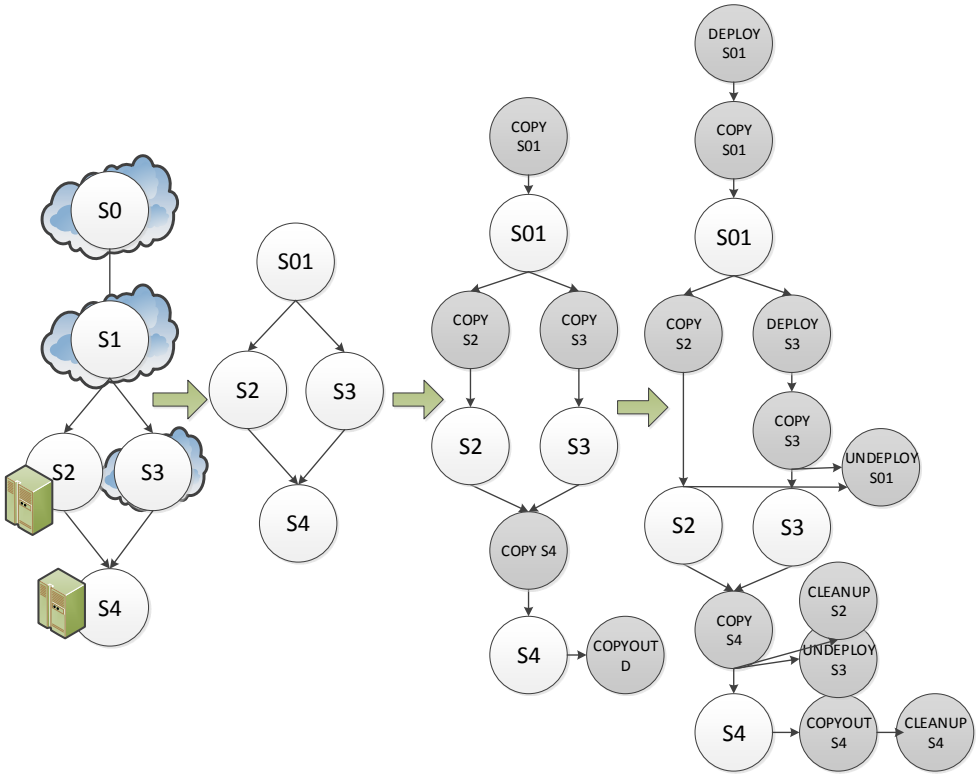

Figure 4.10: Transformation steps of an abstract workflow into an executable workflow. From left to right: 0 (abstract workflow defined by the user), 1 (fusion of stages), 2 (addition of data management stages), 3 (inclusion of stages for cloud computing support).

4.4 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 pseudocode of the workflow execution engine is controlled by two core functions (see Figure 4.11): *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,

```
1: Initialize listStage
2: while nTasks> 0 do
3:   for task= 0 to listStageSize  do
4:     taskStatus←getStatus()
5:     if taskStatus=ENABLED then
6:       taskStatus←RUNNING
7:       runTask()
8:     else
9:       if taskStatus=FINISHED then
10:        nTasks=nTasks−1
11:        listStage.drop(task)
12:        task.enableOutputDependences()
13:      end if
14:    end if
15:  end for
16:  WaitForStatusChangeEvent
17: end while
```

Figure 4.11: Workflow execution engine algorithm

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.

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 [32].

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).

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.

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 4.12. As the compressed input file contains four files, two jobs are submitted for processing two files of *db.zip* each. If after the analysis the token "*(x)*" is not found, then the system considers that the task is not parallel and only one job is submitted in that case, passing all the arguments as parameters to the job.

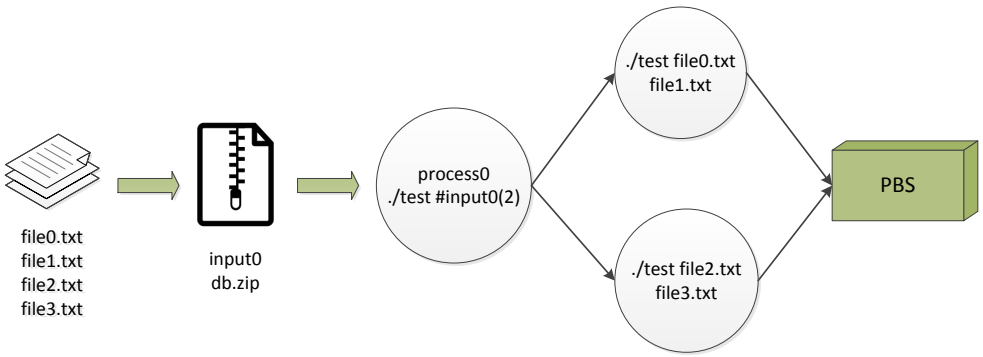

Figure 4.12: Execution of a parallel task.

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).

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.

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.

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.

4.5 Performance optimizations

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

4.5.1 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).

4.5.2 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.

4.5.3 Prefetching: Partial enabling of 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 shown below, in the experimentation section, this functionality is useful for pre-fetching input data to the next stages of the workflow.

4.6 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 database system MongoDB [65]. 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.

4.7 Fault tolerance

In a heterogeneous environment, the failure of a workflow execution can occur for various reasons: the variation in the execution environment configuration (specially in supercomputers and clusters), non-availability of the required software components, overloaded resource conditions, system running out of memory, and faults in computational and network components. A WMS should be able to identify and handle failures and support reliable execution.

As shown in Figure 4.13, workflow failure handling techniques are divided into two different levels, namely *task-level* and *workflow-level*. *Task-level* techniques mask the effects of the execution failure of tasks in the workflow, while *workflow-level* techniques manipulate the workflow structure such as the execution flow to deal with erroneous conditions.

Task-level techniques can be catalogued intro *retry*, *alternate resources*, *checkpoint/restart* and *replication*. The retry technique is the simplest failure recovery technique, as it simply tries to execute the same task on the same resource after failure. The alternate resource technique submits a failed task

to another resource. The checkpoint/restart technique moves failed tasks transparently to other resources, so that the task can continue its execution from the point of failure. The replication technique runs the same task simultaneously on different resources to ensure task executed provided that at least one of the replicas does not fail.

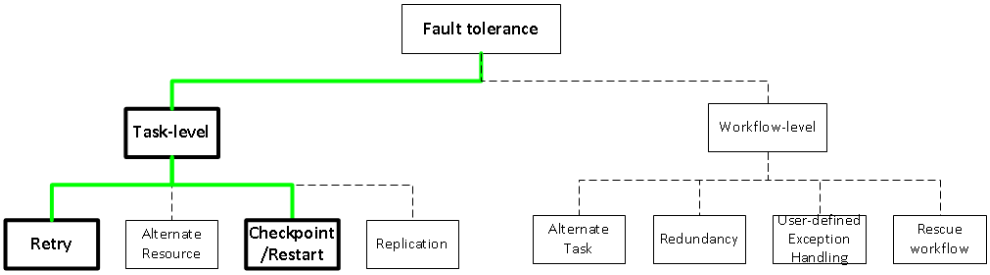

Figure 4.13: Fault tolerance taxonomy.

Workflow-level techniques include *alternate task*, *redundancy*, *user-defined exception handling* and *rescue workflow*. The first three approaches assume that there is more than one implementation for a certain computation with different execution characteristics. The alternate task technique executes another implementation of a certain task if the previous one failed, while the redundancy technique executes multiple alternative tasks simultaneously. The user-defined exception handling allows the users to specify a special treatment for a certain failure of a task in workflow. The rescue workflow technique ignores the failed tasks and continues to execute the remainder of the workflow until no more forward progress can be made. Then, a rescue workflow description, which indicates failed nodes with statistical information, is generated for later submission.

Figure 4.13 depicts that the WMS of this thesis implements a task-level retry based technique. Due to the difference in terms of requirements between the stages that compose a workflow, the WMS defines different retry 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 whole workflow is aborted. The checkpoint/restart technique provided by the persistence module can be always applied if the error

takes place during a deployment stage. In other stages the recovery will only be possible if the sources of all the input files of the stage are still available (i.e. input files of the workflow or outputs that have been replicated to intermediary storage resources via the *replica* field in the JSON specification).

4.8 Provenance

Workflow provenance is a record of the history of the creation of a data object. If the data object was created as the result of a workflow then there must be a way to record the history of that event. The provenance information includes for each process: time stamp, program version number, component version number, execution host, library versions and the data products used. Workflow provenance is crucial for users to be able to follow the evolution of their executions and to determine the cause behind a failure. This information allows users to reproduce the result and reproducibility is a critical component of the scientific method. Because the ad-hoc WMS of this thesis is more a proof-of-concept development than a production system, instead of using the W3C PROV specification [66], it implements a custom and simple provenance module that registers in a local file all the information.

Chapter 5

Use case

The aim of this chapter is to present a realistic comparative genomic workflow used as use case for evaluating the system developed in this thesis. To fully understand the use case, the chapter begins introducing basic terminology related with the use case. After that, the use case is described along with a diagram of the computational steps (or stages) that compose it.

5.1 Preliminary concepts

Prior to describing the workflow, it is necessary to introduce the following concepts:

5.1.1 Comparative genomics

Comparative genomics mainly refers to homology and evolutionary dynamics between organisms, genes and proteins. Be it through complete or specific genomic comparison, such discipline may provide a deeper understanding on how species evolved over time [67]. In addition, functional studies allow for a greater observation on health, phenotype, coding exons, noncoding RNAs and many other aspects related to the species genomic complexity and lineage-specific adaptations [68] [67].

5.1.2 Homology and homology inference

Homology is a very broad comparative genomics concept, which comprises a relationship of common descen between genes or proteins. Even though there are several homology related scenarios, such as orthology, paralogy, horizontal gene transfer, gene loss, xenologs and others - our work focuses on orthology and paralogy [68]. Orthology is characterized when the same genes or proteins are present in distinct species, due to a speciation event. Paralogy relates to duplicated genes - usually in the same species - although they may be inferred in distinct organisms [68].

As ortholog genes tend to preserve their ancestor function, these can be used in order to improve the annotation of data obtained from newly sequenced genes in several organisms. Furthermore, ortholog prediction can also be used to provide better understanding and evolutionary classification of such genes. [68] [69].

High quality ortholog prediction is a desirable aspect for many studies, especially when dealing with incomplete or lacking experimental genomic data [70]. In addition, it also has a direct impact on many comparative genomics tasks, such as functional characterization, genome annotation, conserved regulatory elements identification, orthologous databases creation and others [71] [72] [73] [74].

5.2 Orthosearch

OrthoSearch (Orthologous Gene Searcher) [75] [76] 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 [77] and HMMER [78] which confront an orthologous database with an organism multifasta protein data. The abstract workflow is depicted in Figure 5.1.
It displays that the structure of the Orthosearch pipeline is composed of 8 stages: mafft, fasta2stockholm, hmmbuild, hmmsearch, cat, hmmpress, hmmscan and Reciprocal_Best_Hits.

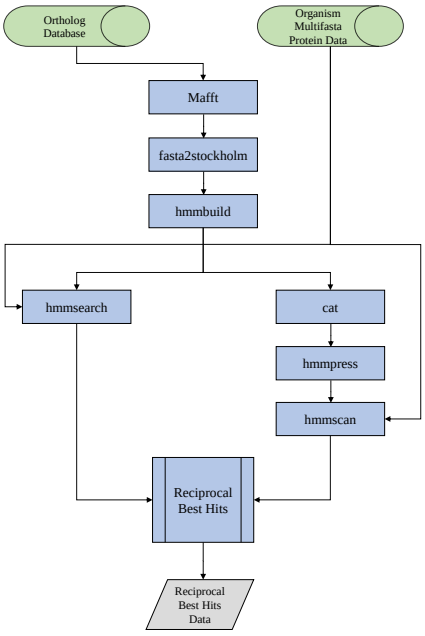

Figure 5.1: Orthosearch abstract workflow.

5.3 Data selection

It was selected a subset of EggNOG database version 4 [79] 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 [80]. In fact, there is no effective treatment or prevention for such infection in humans so far [81].

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 [82] [83].

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 related 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.

Chapter 6

Experiments

This chapter presents the results obtained from the set of experiments carried out with the use case introduced in the previous chapter. On one hand, the aim of the first group of experiments is to evaluate the sequential execution of the use case against the executions on cloud resources with different configuration parameters. On the other hand, the second group of experiments prove the benefits obtained from executing the use case in a heterogeneous environment.

6.1 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 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.

6.2 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 6.1 shows a Gantt chart for the sequential execution of Orthosearch when using the cluster resource while Figure 6.2 the corresponding chart when using the cloud computing asset.

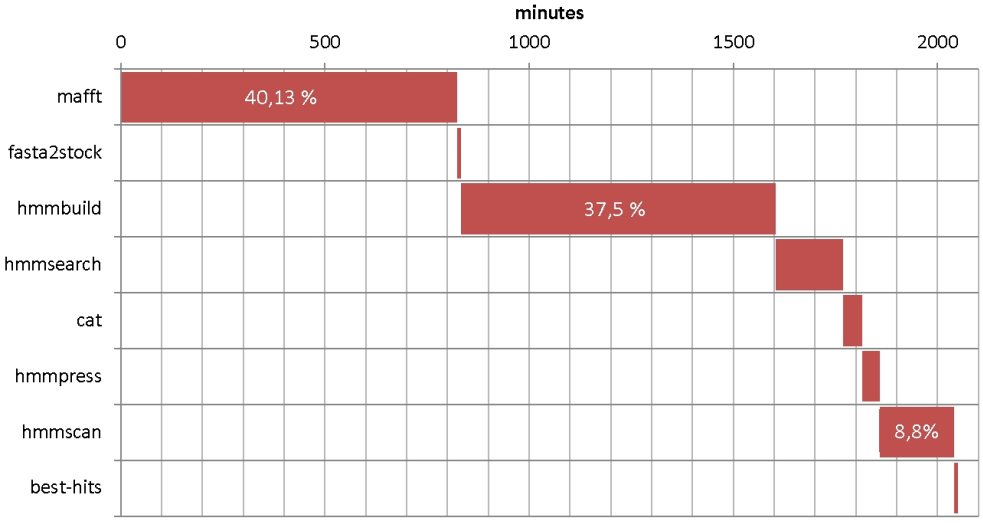

Figure 6.1: Orthosearch serial execution using a cluster.

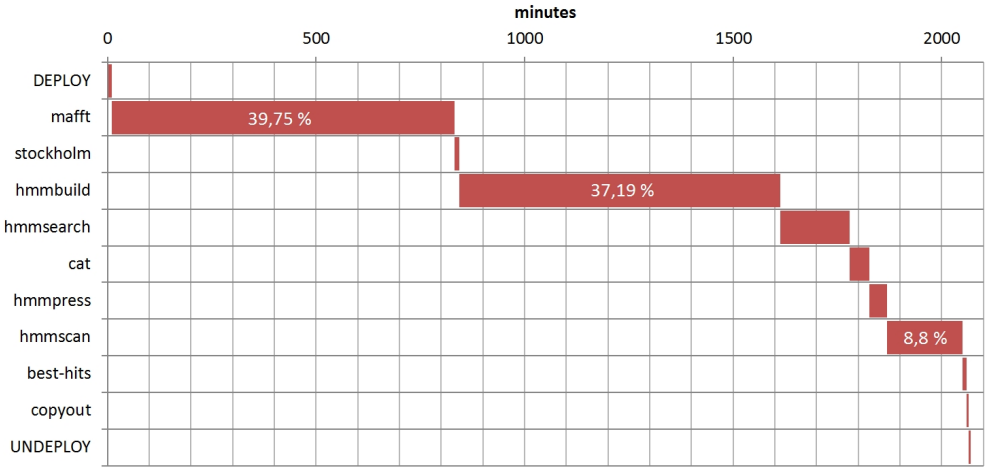

Figure 6.2: Orthosearch serial execution using a Virtual Machine instance.

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.

6.3 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 6.1 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 6.3 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).

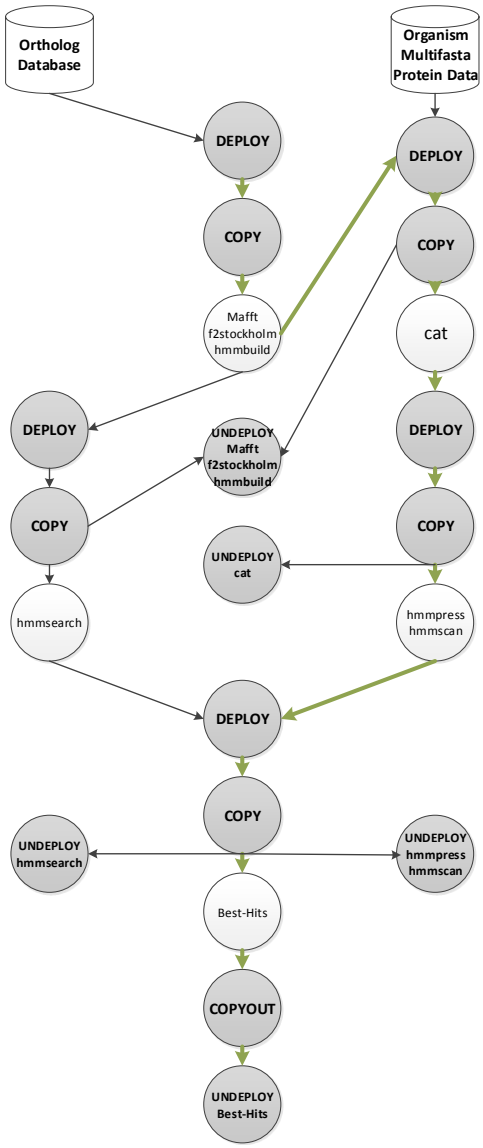

Figure 6.3: Executable workflow for Orthosearch.

Table 6.1: 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

6.3.1 Execution without pre-fetching

Figure 6.4 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.

6.3.2 Execution with pre-fetching

The execution of Orthosearch with the pre-fetching option of the WMS enabled can be seen in Figure 6.5. The main difference with respect to 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.

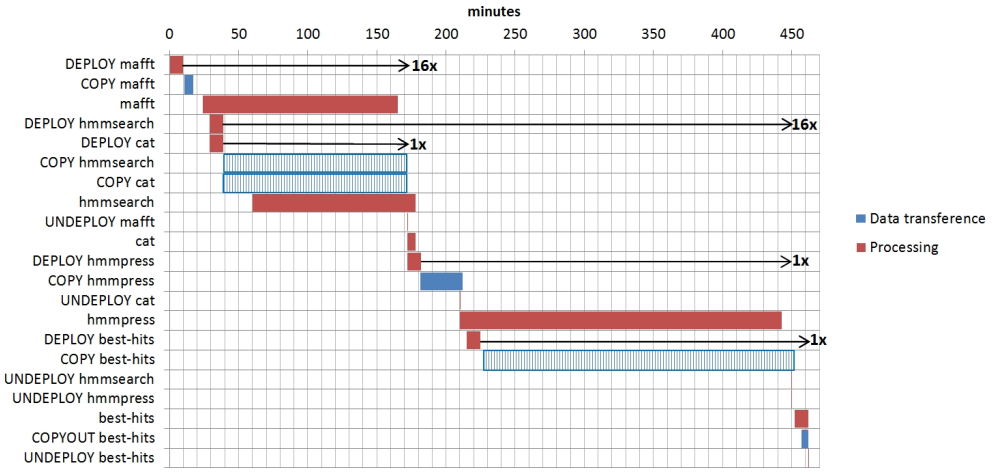

Figure 6.4: Orthosearch execution using the WMS with pre-fetching not enabled.

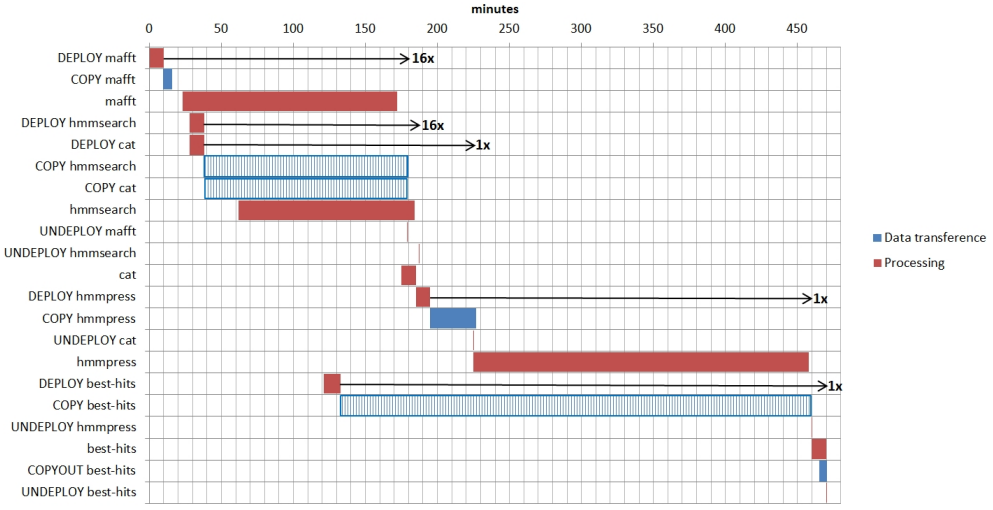

Figure 6.5: Orthosearch execution using the WMS with pre-fetching enabled.

6.3.3 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 experiment 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 UNDEPLOY and the end of DEPLOY), 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.

6.4 Overall analysis

Figure 6.6 and Figure 6.7 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.).

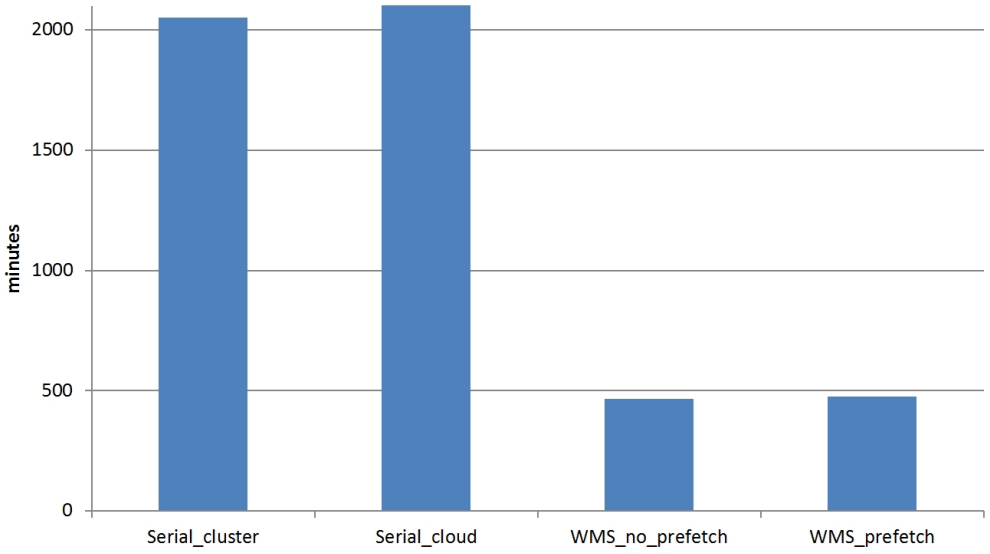

Figure 6.6: Response time comparison for different scenarios.

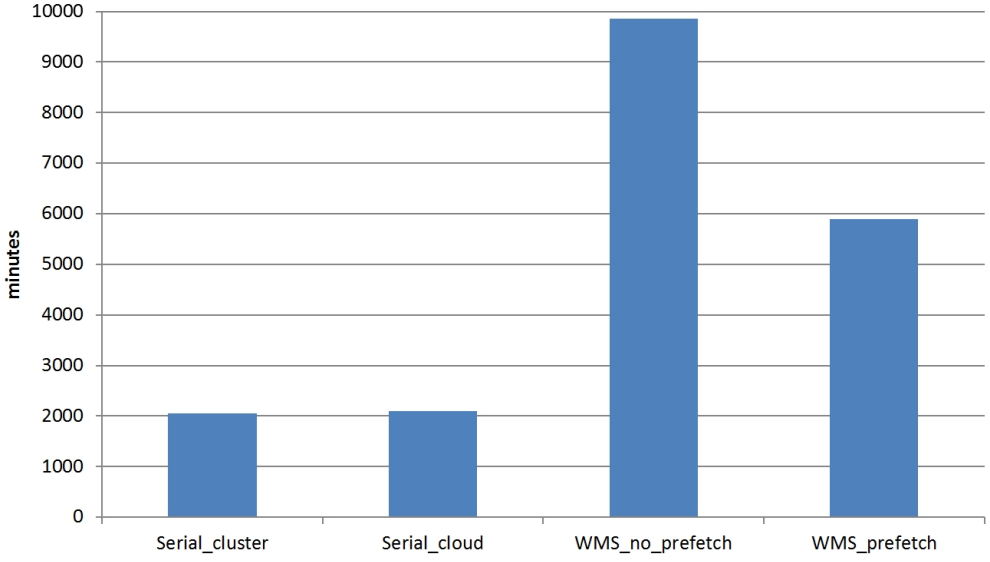

Figure 6.7: CPU usage comparison for different scenarios.

6.5 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.

6.5.1 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. The critical path can be seen in Figure 6.3, 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.

6.5.2 Performance analysis

In Figure 6.8, 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 6.8. 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 one that does not imply additional costs; full

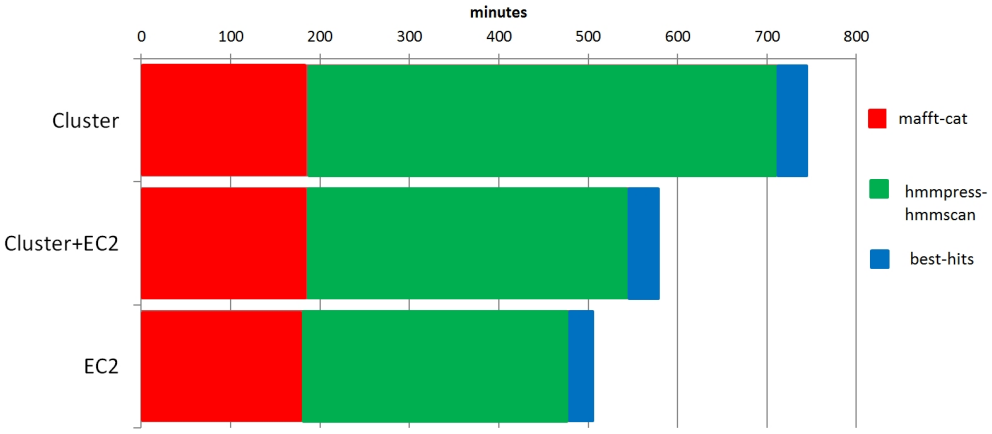

Figure 6.8: Time-line of Orthosearch's critical path for 3 scenarios.

cloud is the fastest one but also the most expensive; and the hybrid scenario is a compromise of both.

Chapter 7

Conclusions and Future Work

7.1 Summary and main contributions

In the e-Science context, 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. 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. Because e-Science processes are modelled with workflows, software components called Workflow Management Systems (WMSs) play a crucial role in this data deluge scenario.

In the last years, 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.

For that reason, the main contribution of this thesis is an ad-hoc solution for managing workflows exploiting the capabilities of cloud computing orchestrators to deploy resources adaptively according to the workload and to combine heterogeneous cloud providers (such as on-premise clouds and public clouds)

and traditional infrastructures (supercomputers and clusters) to minimize costs and response time. The thesis does not propose yet another WMS, but highlights the benefits of the integration of cloud orchestration when running complex workflows. 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 realistic comparative genomics workflow called Orthosearch. An exhaustive analysis has been performed, using several configurations to migrate memory-intensive workload to public infrastructures while keeping other blocks of the experiment running locally. The running time and cost of the experiments is computed and best practices are suggested.

The main contributions of the thesis can be split into three categories: conference contributions and journal contributions and events.

The following list showcases the conference contributions related with this thesis:

- Blanquer Espert, Ignacio; Carrión Collado, Abel Antonio; Hernández García, Vicente; Pignatelli, Miguel; Tamames, Javier. A Comparison Between mpiBLAST on Supercomputers and High-Throughput BLAST on Grid Infrastructures. In: First EELA-2 Conference Bogota (Colombia): CIEMAT; 2009-03-01. 129-137.
- Blanquer Espert, Ignacio; Carrión Collado, Abel Antonio; Hernández García, Vicente; Pignatelli, Miguel; Tamames, Javier. Improving the execution of Bioinformatics applications by using pilot jobs. In: 3RD. IBERIAN GRID INFRASTRUCTURE CONFERENCE VALENCIA: NETBIBLO SL; 2009-05-22. 43-53.
- Blanquer Espert, Ignacio; Carrión Collado, Abel Antonio; Hernández García, Vicente; Conejero Tomas, Vicente; Forment Millet, José Javier. Estimating the horizontal gene transfer between prokaryotes and plants by using e-Science infrastructures. In: Second EELA-2 Conference Choroní (Venezuela): CIEMAT; 2009-11-01. 49-58.
- Blanquer Espert, Ignacio; Carrión Collado, Abel Antonio; Hernández García, Vicente. Estimating the performance of BLAST runs in the EGEE Grid. In: 5th EGEE User Forum, Uppsala (Suecia): European Grid Initiative (EGI); 2010.

- Blanquer Espert, Ignacio; Carrión Collado, Abel Antonio; Hernández García, Vicente. Characterizing Grid experiments in Bioinformatics for an efficient scheduling. In: VECPAR'10, Berkeley, CA (USA): Lawrence Berkeley National Laboratory, CITRIS, University of Porto; 2010.
- Lezzi, D., Rafanell, R., Carrión, A., Espert, I. B., Hernández, V., & Badia, R. M. (2011, August). Enabling e-Science applications on the Cloud with COMPSs. In European Conference on Parallel Processing (pp. 25-34). Springer Berlin Heidelberg.
- Carrión, A., Blanquer, I., & Hernández, V. (2012). A service-based BLAST command tool supported by cloud infrastructures. Stud Health Technol Inform, 175, 69-77.
- Carrión, A., Blanquer, I., Caballer, M., González, C. Y., & Medina, I. (2014). Design of a Generic Architecture for executing Bioinformatics Workflows on Distributed Infrastructures. In IWBBIO (pp. 563-574).
- Carrión, A., Kotowski, N., Caballer, M., Blanquer, I., Jardim, R., & Dávila, A. M. Design and implementation of a Generic and Multi-Platform Workflow System. In 8th Iberian Grid Infrastructure Conference Proceedings (p. 77).
- Carrión, A., Caballer, M., Blanquer, I., Kotowski, N., & Dávila, A. M. R. (2015, January). A Multi-Platform Workflow Management System optimized for Cloud Computing Platforms. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA) (p. 424). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp)

The journal contributions related with the work of this thesis are listed below:

- Avila-George, H., Torres-Jiménez, J., Rangel-Valdez, N., Carrión, A., & Hernández, V. (2012). Supercomputing and grid computing on the verification of covering arrays. The Journal of supercomputing, 62(2), 916-945.
- Himer Avila-George, Jose Torres-Jiménez, Abel Carrión and Vicente Hernández (2012). Using Grid Computing for Constructing Ternary Covering Arrays, Grid Computing - Technology and Applications, Widespread Coverage and New Horizons, Dr. Soha Maad (Ed.), InTech, DOI: 10.5772/36019.
- Abel Carrión, Ignacio Blanquer, Miguel Caballer and Nelson Kotowski (2017). Managing Workflows on top of a Cloud Computing Orchestrator for using heterogeneous environments on e-Science, International Journal of Grid and Web Services, Indersciences, DOI: 10.1504/IJWGS.2017.10003225.

The work related with this thesis and presented in the context of events are:

- Abel Carrión, Ignacio Blanquer and Vicente Hernández. Aplicación biomédica gBLAST. In: 3a Reunión Plenaria Red Espa nola de e-Ciencia, Universidad Politécnica de Valencia (Valencia), 2009.
- Abel Carrión, Ignacio Blanquer and Vicente Hernández. Enabling e-Science applications on the Cloud through VENUS-C. In: Cloud Computing Workshop, UMA (Universidad de Málaga) (Málaga), 2011.
- Abel Carrión, Ignacio Blanquer and Vicente Hernández. Hands-on: Implementation of a BLAST porting to VENUS-C GW and Windows Azure. In: HCCA Cloud Computing Workshop, ICCH - International Co-location Center Hagenberg, Linz (Austria), 2011.

Finally, all the work has been made publicly available in the following GitHub repository: https://github.com/abel-carrion.

7.2 Future Work

There are basically three research lines that can be followed to extend the work developed in this thesis.
The first research line entails adding support of containers to the ad-hoc WMS. In contrast to the infrastructures already supported (cluster, supercomputers and clouds), the complexity of adding the use of containers is trivial because the IM already supports them.

With the container support included and tested, the next logical improvement would be to reuse the computing resources between workflow stages whenever it is possible. In the cloud computing case, the use of the vertical and horizontal elasticity comes in handy to adapt the hardware differences between two consecutive stages. In most cases, this strategy will not only impact by significantly reducing the deployment time of the workflow execution, but also it will minimize the time dedicated to transferring intermediate files.

As it was stated before, the cloud orchestration system is ready to be ported to any state-of-the-art WMS. Thus, the final step would be to integrate all the work done in this thesis and the previous research lines in a mature and in production WMS.

Bibliography

[1] Daniel Atkins. "Revolutionizing science and engineering through cyberinfrastructure: Report of the National Science Foundation blue-ribbon advisory panel on cyberinfrastructure". In: (2003) (cit. on p. 1).

[2] Shannon Bohle. "What is E-science and how should it be managed?" In: *Nature. com, Spektrum der Wissenschaft (Scientific American), http://www. scilogs. com/scientific_and_medicallib raries/what-is-e-science-and-how-should-it-be-managed* (2013) (cit. on p. 1).

[3] Ewa Deelman et al. "Workflows and e-Science: An overview of workflow system features and capabilities". In: *Future Generation Computer Systems* 25.5 (2009), pp. 528–540 (cit. on p. 1).

[4] Christina Hoffa et al. "On the use of cloud computing for scientific workflows". In: *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE. 2008, pp. 640–645 (cit. on pp. 2, 11).

[5] Rodrigo N Calheiros et al. "Adaptive Execution of Scientific Workflow Applications on Clouds". In: *Cloud Computing with e-Science Applications* (2015), p. 73 (cit. on pp. 3, 11).

[6] Naidila Sadashiv and SM Dilip Kumar. "Cluster, grid and cloud computing: A detailed comparison". In: *Computer Science & Education (ICCSE), 2011 6th International Conference on*. IEEE. 2011, pp. 477–482 (cit. on p. 6).

[7] Donald J Becker et al. "BEOWULF: A parallel workstation for scientific computation". In: *Proceedings, International Conference on Parallel Processing*. Vol. 95. 1995 (cit. on p. 6).

[8] David E Culler et al. "Parallel computing on the Berkeley NOW". In: *9th Joint Symposium on Parallel Processing*. 1997 (cit. on p. 6).

[9] *High Performance Virtual Machines*. `http://cseweb.ucsd.edu/groups/csag/html/projects/hpvm.html`. Accessed: 2016-11-15 (cit. on p. 6).

[10] Madhu Chetty and Rajkumar Buyya. "Weaving computational Grids: How analogous are they with electrical Grids?" In: *Computing in Science & Engineering* 4.4 (2002), pp. 61–71 (cit. on p. 7).

[11] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. "A taxonomy and survey of grid resource management systems for distributed computing". In: *Software: Practice and Experience* 32.2 (2002), pp. 135–164 (cit. on p. 7).

[12] Ian Foster. "What is the grid? a three point checklist, July 2002". In: *ThreePoint-Check. pdf* (2006) (cit. on p. 7).

[13] Rajkumar Buyya et al. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility". In: *Future Generation computer systems* 25.6 (2009), pp. 599–616 (cit. on p. 8).

[14] Peter Mell and Tim Grance. "The NIST definition of cloud computing". In: (2011) (cit. on p. 8).

[15] Erwin Laure et al. "Programming the Grid with gLite". In: *Computational methods in science and technology* 12.1 (2006), pp. 33–45 (cit. on p. 10).

[16] Xiaorong Li et al. "Design and development of an adaptive workflow-enabled spatial-temporal analytics framework". In: *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*. IEEE. 2012, pp. 862–867 (cit. on p. 11).

[17] *AWS CloudFormation*. `https://aws.amazon.com/cloudformation/`. Accessed: 2016-11-15 (cit. on p. 13).

[18] *AWS OpsWorks*. `https://aws.amazon.com/opsworks/`. Accessed: 2016-11-15 (cit. on p. 13).

[19] Katarzyna Keahey and Tim Freeman. "Contextualization: Providing one-click virtual clusters". In: *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE. 2008, pp. 301–308 (cit. on p. 13).

[20] Gideon Juve and Ewa Deelman. "Wrangler: virtual cluster provisioning for the cloud". In: *Proceedings of the 20th international symposium on High performance distributed computing*. ACM. 2011, pp. 277–278 (cit. on p. 13).

[21] Daniel Nurmi et al. "The eucalyptus open-source cloud-computing system". In: *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*. IEEE. 2009, pp. 124–131 (cit. on p. 14).

[22] Dejan Milojivic, Ignacio M Llorente, and Ruben S Montero. "Opennebula: A cloud management tool". In: *IEEE Internet Computing* 15.2 (2011), pp. 11–14 (cit. on p. 14).

[23] *HashiCorp: Vagrant (2013)*. `http://www.vagrantup.com/`. Accessed: 2016-11-15 (cit. on p. 14).

[24] *Cloudify*. `http://getcloudify.org/`. Accessed: 2016-11-15 (cit. on p. 15).

[25] Tobias Binz et al. "TOSCA: portable automated deployment and management of cloud applications". In: *Advanced Web Services*. Springer, 2014, pp. 527–549 (cit. on p. 15).

[26] *Chef*. `https://www.chef.io/chef/`. Accessed: 2016-11-15 (cit. on p. 15).

[27] *Puppet*. `https://puppet.com/`. Accessed: 2016-11-15 (cit. on p. 15).

[28] *Ansible*. `http://www.ansible.com/`. Accessed: 2016-11-15 (cit. on pp. 15, 16).

[29] *Heat*. `https://wiki.openstack.org/wiki/Heat`. Accessed: 2016-11-15 (cit. on p. 15).

[30] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: toward an open-source solution for cloud computing". In: *International Journal of Computer Applications* 55.3 (2012) (cit. on p. 15).

[31] Cloud Foundry. *Cloud Foundry*. 2015 (cit. on p. 16).

[32] Miguel Caballer et al. "Dynamic management of virtual infrastructures". In: *Journal of Grid Computing* 13.1 (2015), pp. 53–70 (cit. on pp. 16, 52).

[33] Jose V Carrión et al. "A generic catalog and repository service for virtual machine images". In: *2nd International ICST Conference on Cloud Computing (CloudComp 2010)*. 2010, pp. 1–15 (cit. on p. 16).

[34] Marek Wieczorek, Radu Prodan, and Thomas Fahringer. "Scheduling of scientific workflows in the ASKALON grid environment". In: *ACM SIGMOD Record* 34.3 (2005), pp. 56–62 (cit. on p. 18).

[35] Thomas Fahringer, Jun Qin, and Stefan Hainzer. "Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language". In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Vol. 2. IEEE. 2005, pp. 676–685 (cit. on p. 18).

[36] Rizos Sakellariou and Henan Zhao. "A low-cost rescheduling policy for efficient mapping of workflows on grid systems". In: *Scientific Programming* 12.4 (2004), pp. 253–262 (cit. on pp. 18, 35).

[37] Gabriela Andreea Morar et al. "Meteorological simulations in the cloud with the ASKALON environment". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7640 LNCS (2013), pp. 68–78. ISSN: 03029743. DOI: `10.1007/978-3-642-36949-0\_9` (cit. on p. 18).

[38] Jeremy Goecks, Anton Nekrutenko, and James Taylor. "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences". In: *Genome biology* 11.8 (2010), p. 1 (cit. on pp. 19, 21).

[39] Katherine Wolstencroft et al. "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud". In: *Nucleic acids research* (2013), gkt328 (cit. on p. 19).

[40] Tristan Glatard et al. "Flexible and efficient workflow deployment of data-intensive applications on grids with moteur". In: *International Journal of High Performance Computing Applications* 22.3 (2008), pp. 347–360 (cit. on p. 19).

[41] Ewa Deelman et al. "Pegasus: A framework for mapping complex scientific workflows onto distributed systems". In: *Scientific Programming* 13.3 (2005), pp. 219–237 (cit. on p. 19).

[42] Ewa Deelman et al. "Pegasus in the Cloud: Science Automation through Workflow Technologies". In: *IEEE Internet Computing* 20.1 (2016), pp. 70–76 (cit. on p. 20).

[43] Xiao Liu et al. "SwinDeW-C: a peer-to-peer based cloud workflow system". In: *Handbook of Cloud Computing*. Springer, 2010, pp. 309–332 (cit. on p. 20).

[44] Ian Taylor et al. "The triana workflow environment: Architecture and applications". In: *Workflows for e-Science*. Springer, 2007, pp. 320–339 (cit. on p. 20).

[45] Lavanya Ramakrishnan et al. "VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE. 2009, pp. 1–12 (cit. on p. 20).

[46] Peter Kacsuk et al. "WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities". In: *Journal of Grid Computing* 10.4 (2012), pp. 601–630 (cit. on p. 21).

[47] I Foster et al. *OGSA basic execution service version 1.0*. 2007 (cit. on p. 21).

[48] Ravi Madduri et al. "The Globus Galaxies platform: delivering science gateways as a service". In: *Concurrency and Computation: Practice and Experience* 27.16 (2015), pp. 4344–4360 (cit. on p. 21).

[49] Michael Wilde et al. "Swift: A language for distributed parallel scripting". In: *Parallel Computing* 37.9 (2011), pp. 633–652 (cit. on p. 21).

[50] Daniel de Oliveira et al. "Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows". In: *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE. 2010, pp. 378–385 (cit. on p. 21).

[51] Abel Carrión et al. "Design of a Generic Architecture for executing Bioinformatics Workflows on Distributed Infrastructures." In: *IWBBIO*. 2014, pp. 563–574 (cit. on p. 29).

[52] Jia Yu and Rajkumar Buyya. "A taxonomy of workflow management systems for grid computing". In: *Journal of Grid Computing* 3.3-4 (2005), pp. 171–200 (cit. on p. 29).

[53] Wil MP van Der Aalst et al. "Workflow patterns". In: *Distributed and parallel databases* 14.1 (2003), pp. 5–51 (cit. on p. 32).

[54] Wil MP van der Aalst et al. "Advanced workflow patterns". In: *International Conference on Cooperative Information Systems*. Springer. 2000, pp. 18–29 (cit. on p. 32).

[55] Ewa Deelman et al. "Workflow management in GriPhyN". In: *Grid Resource Management*. Springer, 2004, pp. 99–116 (cit. on p. 34).

[56] Ewa Deelman et al. "Pegasus: Mapping scientific workflows onto the grid". In: *Grid Computing*. Springer. 2004, pp. 11–20 (cit. on pp. 34, 48).

[57] Bertram Ludascher, Ilkay Altintas, and Amarnath Gupta. "Compiling abstract scientific workflows into web service workflows". In: *Scientific*

and Statistical Database Management, 2003. 15th International Conference on. IEEE. 2003, pp. 251–254 (cit. on p. 34).

[58] Kaizar Amin et al. "Gridant: A client-controllable grid workflow system". In: *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*. IEEE. 2004, 10–pp (cit. on p. 34).

[59] Tim Bray et al. "Extensible markup language (XML)". In: *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210* 16 (1998), p. 16 (cit. on p. 35).

[60] Francisco Hernández et al. "A graphical modeling environment for the generation of workflows for the globus toolkit". In: *Component Models and Systems for Grid Applications*. Springer, 2005, pp. 79–96 (cit. on p. 35).

[61] Tadao Murata. "Temporal uncertainty and fuzzy-timing high-level Petri nets". In: *International Conference on Application and Theory of Petri Nets*. Springer. 1996, pp. 11–28 (cit. on p. 35).

[62] Grady Booch. *The unified modeling language user guide*. Pearson Education India, 2005 (cit. on p. 35).

[63] *JavaScript Object Notation*. `http://json.org/`. Accessed: 2016-11-15 (cit. on p. 36).

[64] *Jackson:High-performance JSON processor*. `http://jackson.codehaus.org/`. Accessed: 2016-11-15 (cit. on p. 47).

[65] *MongoDB*. `http://www.mongodb.org/`. Accessed: 2016-11-15 (cit. on p. 57).

[66] Paolo Missier, Khalid Belhajjame, and James Cheney. "The W3C PROV family of specifications for modelling provenance metadata". In: *Proceedings of the 16th International Conference on Extending Database Technology*. ACM. 2013, pp. 773–776 (cit. on p. 59).

[67] Ross C Hardison. "Comparative genomics." In: *PLoS biology* 1.2 (Nov. 2003), E58. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.0000058 (cit. on p. 61).

[68] Eugene V Koonin. "Orthologs, paralogs, and evolutionary genomics." en. In: *Annual review of genetics* 39 (Jan. 2005), pp. 309–38. ISSN: 0066-4197. DOI: 10.1146/annurev.genet.39.073003.114725 (cit. on pp. 61, 62).

[69] Eugene V Koonin and Michael Y Galperin. *Sequence - Evolution - Function*. en. 2003 (cit. on p. 62).

[70] Christophe Dessimoz et al. "Toward community standards in the quest for orthologs." In: *Bioinformatics (Oxford, England)* 28.6 (Mar. 2012), pp. 900–4. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/bts050 (cit. on p. 62).

[71] Rafael R C Cuadrat et al. "An orthology-based analysis of pathogenic protozoa impacting global health: an improved comparative genomics approach with prokaryotes and model eukaryote orthologs." en. In: *Omics : a journal of integrative biology* 18.8 (Aug. 2014), pp. 524–38. ISSN: 1557-8100. DOI: 10.1089/omi.2013.0172 (cit. on p. 62).

[72] Debra L Fulton et al. "Improving the specificity of high-throughput ortholog prediction." In: *BMC bioinformatics* 7.1 (Jan. 2006), p. 270. ISSN: 1471-2105. DOI: 10.1186/1471-2105-7-270 (cit. on p. 62).

[73] Christophe Dessimoz. "Editorial: Orthology and applications." In: *Briefings in bioinformatics* 12.5 (Sept. 2011), pp. 375–6. ISSN: 1477-4054. DOI: 10.1093/bib/bbr057 (cit. on p. 62).

[74] Li Li, Christian J Stoeckert, and David S Roos. "OrthoMCL: identification of ortholog groups for eukaryotic genomes." In: *Genome research* 13.9 (Sept. 2003), pp. 2178–89. ISSN: 1088-9051. DOI: 10.1101/gr.1224503 (cit. on p. 62).

[75] Sergio Manuel Serra da Cruz et al. "OrthoSearch". In: *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08*. New York, New York, USA: ACM Press, Mar. 2008, p. 1282. ISBN: 9781595937537. DOI: 10.1145/1363686.1363983 (cit. on p. 62).

[76] Sérgio Manuel Serra da Cruz et al. "Detecting distant homologies on protozoans metabolic pathways using scientific workflows." In: *International journal of data mining and bioinformatics* 4.3 (Jan. 2010), pp. 256–80. ISSN: 1748-5673 (cit. on p. 62).

[77] Kazutaka Katoh and Daron M Standley. "MAFFT multiple sequence alignment software version 7: improvements in performance and usability." In: *Molecular biology and evolution* 30.4 (Apr. 2013), pp. 772–80. ISSN: 1537-1719. DOI: 10.1093/molbev/mst010 (cit. on p. 62).

[78] Robert D Finn, Jody Clements, and Sean R Eddy. "HMMER web server: interactive sequence similarity searching." In: *Nucleic acids research* 39.Web Server issue (July 2011), W29–37. ISSN: 1362-4962. DOI: 10.1093/nar/gkr367 (cit. on p. 62).

[79] Sean Powell et al. "eggNOG v4.0: nested orthology inference across 3686 organisms." In: *Nucleic acids research* 42.Database issue (Jan. 2014), pp. D231–9. ISSN: 1362-4962. DOI: 10.1093/nar/gkt1253 (cit. on p. 63).

[80] Mitchell S Abrahamsen et al. "Complete genome sequence of the apicomplexan, Cryptosporidium parvum." In: *Science (New York, N.Y.)* 304.5669 (Apr. 2004), pp. 441–5. ISSN: 1095-9203. DOI: 10.1126/science.1094786 (cit. on p. 63).

[81] Ping Xu et al. "The genome of Cryptosporidium hominis." In: *Nature* 431.7012 (Oct. 2004), pp. 1107–12. ISSN: 1476-4687. DOI: 10.1038/nature02977 (cit. on p. 63).

[82] Michael P Barrett and Simon L Croft. "Management of trypanosomiasis and leishmaniasis." In: *British medical bulletin* 104 (Jan. 2012), pp. 175–96. ISSN: 1471-8391. DOI: 10.1093/bmb/lds031 (cit. on p. 63).

[83] Wangeci Gatei et al. "Cryptosporidiosis: prevalence, genotype analysis, and symptoms associated with infections in children in Kenya". In: *Am J Trop Med Hyg* 75.1 (July 2006), pp. 78–82 (cit. on p. 63).

List of Figures

List of Tables