Universidad Zaragoza
1542

Trabajo Fin de Máster

Integración dinámica de entornos de computación heterogéneos para la ejecución de *workflows* científicos

Autor

Sergio Hernández de Mesa

Director

Pedro Álvarez Pérez-Aradros

Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas
Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE)

Junio 2012

Repositorio de la Universidad de Zaragoza – Zaguan http://zaguan.unizar.es

Anexo A | *A Framework for the Flexible Deployment of Scientific Workflows in Grid Environments*

En este Anexo se muestra el artículo titulado "*A Framework for the Flexible Deployment of Scientific Workflows in Grid Environments*" el cual ha sido aceptado para su publicación en la *III International Conference on Cloud Computing, GRIDs, and Virtualization* (CLOUD COMPUTING 2012) que se celebrará en Niza (Francia), del 22 al 27 de Julio de 2012.

El artículo se presenta manteniendo el formato con el que aparecerá en el congreso mencionado.

A Framework for the Flexible Deployment of Scientific Workflows in Grid Environments

Javier Fabra, Sergio Hernández, Pedro Álvarez, Joaquín Ezpeleta
Aragón Institute of Engineering Research (I3A)
Department of Computer Science and Systems Engineering
University of Zaragoza, Spain
Email: {jfabra,shernandez,alvaper,ezpeleta}@unizar.es

***Abstract*—Scientific workflows are generally programmed and configured to be executed by a specific grid-based system. The integration of heterogeneous grid computing platforms in order to build more powerful infrastructures and the flexible deployment and execution of workflows over them are still two open challenges. Solutions based on meta-scheduling have been proposed, but more flexible and decentralized alternatives should be considered. In this paper an alternative framework based on the use of a tuple-based coordination system and a set of mediation components is proposed. This framework provides users with scalability and extensibility mechanisms, being suitable for a wide variety of scenarios. As a use case, the First Provenance Challenge has been implemented using two different workflow technologies executed over the framework, Nets-within-Nets and Taverna, and transparently deployed on two different computing insfrastructures.**

***Keywords* – middleware for integration, scientific workflow deployment, grid-based systems.**

I. Introduction

Grid computing emerged as a paradigm for the development of computing infrastructures able to share heterogeneous and geographically distributed resources [1]. Due to their computational and networking capabilities, this type of infrastructure has turned into execution environments suitable for scientific workflows. Scientific workflows are a type of workflow characterized for being composed by a large number of activities whose execution requires a high computation intensity and complex data management.

Currently, many efforts are being carried out in the field of scientific computing to execute their experiments taking full advantage of grid technologies. Two important open challenges in this area are the integration of heterogeneous grid computing platforms in order to build more powerful infrastructures and the flexible deployment and execution of workflows over them. Some authors have proposed solutions based on the use of meta-schedulings without considering dynamic behaviours or workloads. However, in order to tackle with the nature of grids, it is required to consider more flexible and decentralized alternatives.

In this paper, a framework able to tackle the previous challenges is proposed. As shown in [2], [3], the use of a broker based on the Linda coordination model [4] and a set of mediators facilitates the flexible integration of heterogeneous grid computing environments, addressing the challenge of creating more powerful infrastructures. These components encapsulate and handle specific features of various computing environments integrated into our framework, being programmers unaware of this heterogeneity. As a result, the tasks that compose a workflow can be executed in a flexible way using different computing environments. Unlike current proposals the framework is not based on the use of a meta-scheduler to perform global scheduling decisions, but each computing environment competes to execute jobs according to the availability of its own grid resources. In order to implement this alternative scheduling model, each one of these computing environments is represented in the broker by a specific mediator able to achieve suitable scheduling decisions. Hybrid computing environments could be easily integrated implementing new mediators. On the other hand, scientific workflows can be programmed independently of the execution environment in which they will be executed. The Net-within-Nets paradigm [5] and the Renew tool [6] have been used for programming this type of workflows. This is also compatible with other existing workflow programming languages. Indeed, Taverna workflows can be programmed using the framework services or translated to our programming language and then executed.

The remainder of the paper is organized as follows. Section II introduces some related work. In Section III, the architecture of the framework is presented. The role of the Linda-based broker, its implementation details and task dispatching mechanisms are described in Section IV. The flexible integration of heterogenous grid middlewares and grid management components with the broker is then detailed in Section V. The features and new capabilities are shown by means of an example that implements the First Provenance Challenge in Section VI. Finally, conclusions are depicted in Section VII.

II. Related work

A considerable progress has been made in the understanding of the particular nature of scientific workflows and the implementation of grid-based systems for their specification, scheduling, and execution. A detailed survey of existing grid workflow systems is presented in [7], [8]. The comparison of several systems shows relevant differences in the building and execution of workflows that causes experiments programmed by scientists and engineers to be strongly coupled to the underlying grid-based execution system. This coupling forces

grid administrators to perform relevant configuration and integration efforts in most of the scientific workflow deployments. Therefore, some interesting challenges are still open: the ability to program scientific workflows independently of the execution environment, the portability of scientific workflows from one execution environment to another, or the integration of heterogeneous execution environments to create more powerful computation infrastructures, for instance. Consequently, research efforts should concentrate on the definition of new high-level programming constructs independent of specific grid technologies and also on the provision of execution infrastructures able to interface multiple providers. This type of infrastructure should integrate software adaptation layers for translating generic management operations to provider-specific APIs. Additionally, new strategies of resource brokering and scheduling should be integrated into these execution environments to facilitate the utilization of multiple-domain resources and the allocation and binding of workflow activities to them.

Let us briefly resume some of the current proposals for provisioning flexible and extensible execution infrastructures. On the one hand, different grid-based systems built on a *meta-scheduler* have been proposed [9], [10], [11]. A meta-scheduler is a middleware component that provides advanced scheduling capabilities on a grid consisting of different computing platforms. The software architecture of all these solutions is very similar and is composed of the following components: a resource monitoring system to collect information from integrated computing platforms, a meta-scheduler to distribute jobs among grid resources using different scheduling policies [12] and, finally, a set of adaptation components to achieve mediation between middleware components and computing platforms. On the other hand, architectures based on the integration of meta-schedulers have been adapted for taking advantage of Cloud technologies [13], [14], [11]. Resulting computing environments comprise of virtualized services usage-based payment models in order to achieve more efficient and flexible solutions, where the supported functionality will be no longer fixed or locked to underlying infrastructure.

III. An Open Framework for Programming and Executing Scientific Workflows

In short, the main goals of our approach are:

- To execute scientific workflows programmed using a High-level Petri nets formalism or other standard languages widely accepted by the scientific community.
- To simultaneously work with different and heterogeneous grid middlewares or with middlewares implemented using different technologies (e.g. Web services). At this respect, workflow execution engines must be uncoupled from specific grid technologies.
- To allow the addition or removal of resources without previous announcement.
- To support different scheduling strategies and policies in the execution environment. The use of a particular scheduling strategy or policy should depend on the characteristics and requirements of each workflow application.

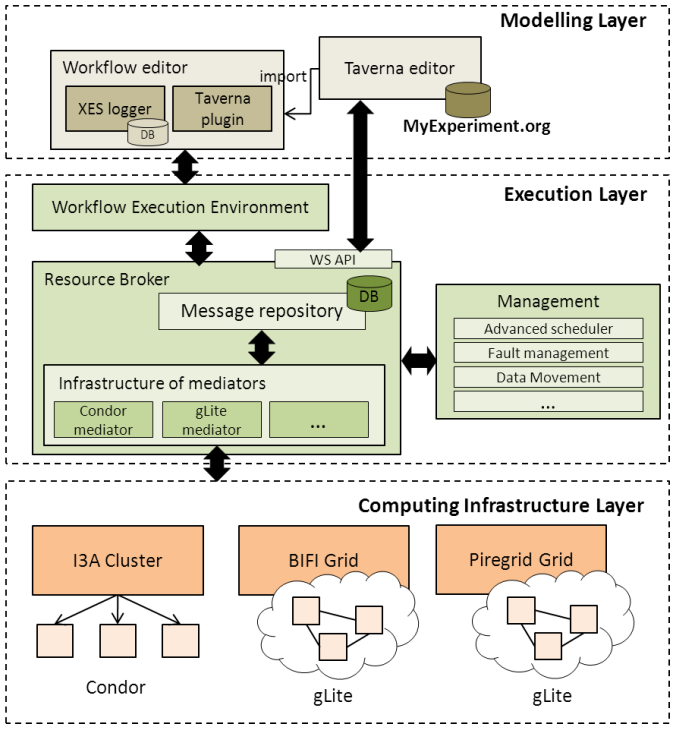

Fig. 1. Architecture of the execution environment.

Figure 1 shows the high-level architecture of the proposed framework. As shown, the architecture consists of three layers: the *modelling layer*, the *execution layer* and the *computing infrastructure layer*. In the following, each layer as well as its main components and interfaces are described in detail.

Firstly, the *modelling layer* consists of a set of tools for the programming of workflow applications. A workflow can be developed using the broker services, which are exposed through its Web service interface, using a workflow modeling tool such as Taverna [15], for instance. Also, we propose the use of Reference nets, a subclass of Petri nets, to implement workflow applications from the perspective of the Nets-within-Nets paradigm [5]. Nevertheless, other high-level programming languages for workflows could be also used by scientific communities (e.g. physicists, biologists or astronomers) for programming their workflows. With respect to this issue, plugins can be added to the modelling layer to support existing or new modelling approaches, such as the Taverna plugin shown in Figure 1, for instance. This plugin allows to import workflows programmed with Taverna, which are automatically translated to the workflow format in the workflow editor and then directly executed. A good repository for these type of workflows is the scientific community hosted at *MyExperiment.org*. In this work, Renew [6] is used as a workflow editor. Renew is an academic open-source tool that allows the direct execution of Reference nets without any additional coding process and which represents a worth benefit for the final user.

Secondly, the *execution layer* is composed of the core components. The *workflow execution environment* is responsible for controlling the execution of workflows and submitting tasks to the *resource broker* when they must be executed. Internally, the broker consists of a *message repository* and a set of *mediators*. Messages are used to encapsulate any information that is passed through the components of the system. A message can describe a task to be executed or the result of its execution, for instance. Mediators encapsulate the heterogeneity of a specific grid middleware, having a complete knowledge of its capabilities. This knowledge is used for making dispatching decisions (which specific computing infrastructure will execute a pending task?). Subsequently, the grid middleware of the selected computing platform will schedule the set of resources needed for executing the task. As a result, the broker uncouples the workflow execution environment from the specific details about the grid-based computing infrastructures where tasks will be executed. This design avoids the need for a close integration of the workflow execution environment with specific grid middlewares used for the execution of tasks.

Let us now go deeper into the description of the two components of the broker. On the one hand, the Linda coordination model [4] has inspired the implementation of the message repository. Messages are encoded as tuples and stored into a tuple space. The interface of the repository provides a set of operations for accessing the tuples stored in the tuple space according to the semantics of Linda. In Section IV we will depict the advantages of using a Linda-based repository and provide details about its implementation. On the other hand, *mediators* are required for achieving the aforementioned uncoupled integration. In general, a mediator is an entity that directly communicates with the tuple repository, matches and retrieves special-tagged tuples and processes them. In our approach, each grid middleware is represented by a mediator. Internally, this mediator is responsible for: i) having a complete information of the grid resource it represents; ii) interacting with the tuple repository to find at run-time tasks that could be executed by the set resources of its middleware; iii) dispatching the task to the middleware for its execution and controlling the input and output data transference; and, finally, iv) storing the results of the executed task in the tuple repository as tuples. Mediators of different and heterogeneous grid middlewares could compete for the execution of a specific task. Currently, as it will be described in Section V, different mediators have been implemented for the grid middleware we have access to (Condor and gLite) and then integrated into the *infrastructure of mediators*.

On the other hand, a set of *management components* has also been integrated into the execution layer to support the execution of workflow applications: the fault management component, the data movement component or the advanced scheduling component, for instance. The integration procedure of these components is similar to the one used by mediators. A management component interacts with the tuple repository in order to match and retrieve special-tagged tuples and then processes them. Therefore, the action of these components can be triggered as a result from the previous processing, which allows to dynamically compose complex action chains. In Section V the component for the fault management subsystem and its integration will be detailed.

Finally, the *computing infrastructure layer* is composed of different and heterogeneous computing platforms. The interaction with these platforms is managed by the corresponding grid middlewares. Currently, three computing platforms are integrated in the framework we manage: the HERMES cluster hosted by the Aragón Institute of Engineering Research (I3A, http://i3a.unizar.es/), which is managed by the Condor middleware; and the two research and production grids managed by the gLite middleware and hosted by the Institute for Biocomputation and Physics of Complex Systems (BIFI, http://bifi.es/en/) belonging to the European Grid Initiative (EGI, http://www.egi.eu/), namely AraGrid (http://www.aragrid.es/) and PireGrid (http://www.piregrid.eu/).

To sum up, the open nature of the proposed solution is provided by the resource broker, composed of a Linda-based repository and a set of mediators, providing scientists with a high level of abstraction and flexibility when developing workflows. On the one hand, workflow programmers must concentrate on the functional description of workflow tasks and corresponding involved data. Specific details about the computing platforms where these tasks will be executed are ignored from the programmer perspective. On the other hand, the message repository facilitates the integration of mediators and management components and the scalability of the overall framework. Currently, its dispatching model is based on the functional capabilities of the computing platforms managed by the set of mediators. And, finally, these mediators are responsible for encapsulating the technological heterogeneity of the different types of grid middlewares and resource-access technologies (e.g. Web services). New mediators may be easily added in order to integrate new middlewares or technologies.

IV. Linda-based task dispatching

As previously stated, the resource broker is composed of a message repository and a set of components (mediators) that interact through this space by means of the exchange of messages. In this section, the role of the Linda-based message repository and the corresponding task description and dispatching mechanisms are presented.

Linda [4] is a coordination model based on two notions: tuples and a tuple-space. A tuple is something like [”Gelernter”, 1989], a list of untyped values. The tuple space is a collection of tuples stored in a shared and global space that can be accessed with certain operations, that allow processes to read and take tuples from and write them into it in a decentralized manner. For instance, the operation `in(x,["Gelernter", ?])` tries to match the *template* `["Gelernter", ?]`, which contains a wildcard, with a tuple in the shared space. If there is a match, a tuple is extracted from the tuple space and assigned to variable `x`; otherwise, the process blocks until a

matching tuple appears. The matching is free for the wildcard, but literal for constant values. The Linda matching mechanism allows easily programming distributed synchronization processes.

Linda-based coordination systems have been widely used for communicating and coordinating distributed processes. Their success in distributed systems is due to a reduced set of basic operations, a data-driven coordination and a space and time uncoupled communication among processes that can cooperate without adapting or announcing themselves [16].

Let us now introduce how tuples are describe and dispatched in our appraoch. Tuples are used to code the information needed for submitting a job to a grid middleware or recovering the result (or an exception) of an executed job. A tuple structure based on the *Job Submission Description Language* standard, JSDL [18], has been adopted. From the job submission point of view, this representation includes the specification of the application to be executed, the references to input and output data (represented by the corresponding URIs), a description of the host required for its execution (operating system, CPU architecture and features, memory, network bandwidth, etc.), QoS parameters and, optionally, the grid middleware responsible for its execution. In case the target grid platform is not specified, different mediators compete for the job execution in base to certain policies. On the other hand, a result tuple contains a reference to the original request, a reference to the output data and the execution log (grid and host used for the job execution, execution costs and QoS results, mainly). If an error occurs, the result tuple will contain the information about it. The fault handling component, which handles these faults, will be depicted in Section V.

Once the tuple representing a job has been created, the workflow execution environment puts it into the message repository by means of an `out` operation. Each grid computing platform is connected to the platform by means of a mediator, which knows the applications that could be locally executed by its grid and the description of the available internal resources. Each mediator is then waiting for tuples that encode such job requests able to be executed by its grid. Obviously, this waiting will depend on the availability at run-time of the grid and its capabilities. An `in` operation is invoked by the mediator in order to retrieve a tuple of its interest, using the Linda matching mechanism. Then, the retrieved tuple is locally processed by the mediator to perform the corresponding invocation to the grid middleware it represents.

If many grid computing platforms are able to execute a job, their mediators will compete to retrieve the job request tuple. The Linda matching mechanism is non-deterministic and, therefore, it does not offer any further guidance about which mediator will retrieve the job request tuple. In this work, the use of WS-PTRLinda, an extension of a previous distributed Linda-based implementation of a message broker, called *DRLinda* [17], is proposed. As DRLinda, WS-PTRLinda was developed using Nets-within-Nets and the Renew tool, the same technologies we used for programming workflow applications. WS-PTRLinda provides a new Web-service based interface (SOAP 1x. SOAP2 and REST), support for persistence of the tuple space (for high-availability demanding environments), and a timeout mechanism useful for failure detection. Currently, a basic and non-deterministic scheduling is being used for dispatching job requests to grid mediators. In [17] we proposed and implemented some alternative matching mechanisms to solve specific problems. Similarly, new grid-oriented matching mechanisms could be defined to extend the scheduling policies of the broker (e.g. a QoS-based scheduling policy). Let us finally comment on two relevant advantages of this Linda-based brokering. Firstly, the cooperation is uncoupled because the execution environment does not have any prior knowledge about mediators and vice versa. The interaction style is adequate enough to be used in environments where it is very important to reduce as much as possible the shared knowledge between different components. Also, writing and reading components can cooperate without adapting or announcing themselves. New mediators could be added/removed without affecting the rest of components integrated into the framework.

V. Flexible integration of grid middlewares

Following the presented approach, different types of resources and components (execution engines, management components or mediators, for instance) can be integrated in an easy and uncoupled way. The only requirement for these components is to implement the Linda coordination API in order to put and remove tuples. Besides, components can be added or removed dynamically and transparently to the rest of the system, facilitating this way the scalability and adaptation of the framework.

In this section two different types of integrated components are presented. The first one is a mediator able to interact with the Condor middleware, whereas the second one is a fault management component. When a fault is detected during the execution of a job, this component will re-schedule the job according to different policies. Our aim is to illustrate how this solution is able to interact with grid computing platforms managed by heterogeneous grid middlewares.

A. Interaction with the Condor middleware

As previously described, the framework is able to interact with several underlying grid infrastructures. Let us depict how a mediator has been developed to integrate a Condor middleware. Specifically, this mediator is responsible for the interaction with the HERMES cluster. Figure 2 shows the functional components of the mediator required for supporting such interaction. Additionally, this mediator can be reused for interacting with any computing platform managed by Condor.

The *Job Manager* interacts with the Linda-based broker depicted in the previous section in order to read job requests and write their results. Obviously, all request types that could be fulfilled by the cluster must be known by the manager. For this purpose, the *Internal Resource Registry* knows the list of applications that could be locally executed and the description of available internal resources. This registry should

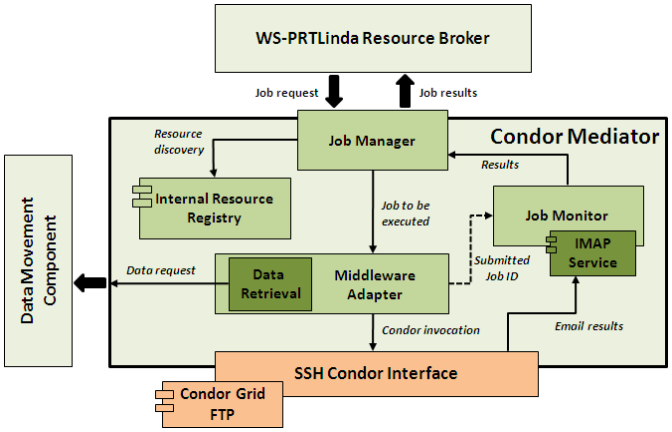

Fig. 2. Components of the Condor mediator.

monitor the cluster and dynamically update its information, but at this first implementation of the Condor mediator this information is static. Once a job request has been retrieved, the manager sends it to the *Middleware Adapter* component that is responsible for translating the request into a Condor job. Before submitting the job to the cluster via the SSH protocol, the adapter internally carries out two important tasks. First, it assigns an identifier to the job (*Job ID*) and sends it to the *Job Monitor* component. This ID will be used to correlate jobs and tuples. In case the input data required by a job are stored in an external computing platform, the adapter interacts with the Data Movement component for moving them (or making a copy) into the Condor cluster. After that, the adapter submits the job to the Condor middleware.

Internally, Condor can schedule the execution of submitted jobs depending on the local state of its resources. The goal is to achieve the best possible throughput. Therefore, a double scheduling can be done in the approach, similarly to the hierarchical scheduling model described in [19]. Once the job execution has been completed, results are sent through a logging mechanism (in our case, SMTP-IMAP) service integrated in the *Job Monitor*. This component maps received results with job requests and forwards them to the job manager. Finally, results are written in the broker so they can be then taken by the workflow application that submitted the original request.

This design and implementation is quite flexible and provides reusability. For instance, we have also developed a mediator to interact with the gLite middleware used in AraGrid. Its design is similar to the previous one. In fact, most internal components have been reused, as the job manager and the internal resource registry, and others components have been adapted, as the middleware adapter or the job monitor, for instance.

B. Fault handling

When dealing with scientific workflows, failures can arise at several levels. In this work, we will focus on those faults and exceptions that happen at the execution level. When the execution of a job fails, the corresponding mediator captures the fault and puts an error tuple into the message repository. This tuple, which will be processed by the *Fault management component*, contains information about the cause of the fault that will be used by the manager to take a decision with respect to the job execution. Different decisions could be taken: to submit the job again to the same grid computing platform, to submit the job to an alternative and reliable grid computing platform or to notify the error to the workflow executing environment in case the error persists, for instance. In the last case, most grid solutions offer two different ways to manage the fault: corrective actions or alternative workflows.

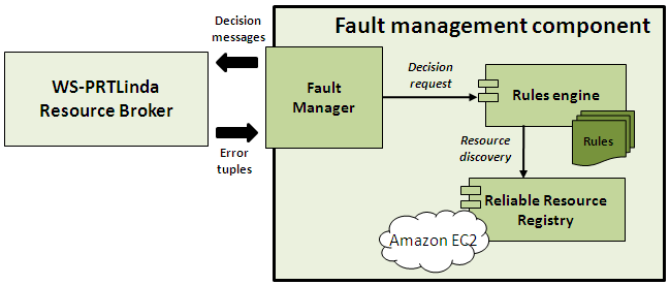

Fig. 3. Components of the fault management component.

Figure 3 shows the internal design of the fault management component. A *Fault Manager* interacts with the message repository in order to retrieve error tuples and to write the corresponding decision tuple. When an error tuple is found, the fault manager processes it and creates a decision request that is sent to a decision maker. We have used a *rules engine* as the decision maker. Rules are encoded in RuleML (the standard Web language for rules using XML markup [20]) and describe the corrective actions that will be executed in case of each type of error. These actions can be changed and modified at runtime, providing adaptation capabilities based on specific scenarios. Normally, the job will be sent again for a new execution on the corresponding infrastructure. However, in case it fails again or even if the error tuple contains some critical information, a usual action is to send the job request to a reliable grid middleware (our ultimate goal is the successful execution of job requests). Reliable grid middlewares have special characteristics (number of nodes, throughput, rejection rate, etc.), which turn them into more suitable candidates for a difficult job execution. For this purpose, a *Reliable Resource Registry* has been implemented and integrated in the fault management component. The current version of the registry contains a list of reliable grid middlewares. This list is used by the rules engine to decide in which middleware the failed job request will be executed. Finally, the fault manager puts a new job request tuple into the broker, specifying the grid middleware responsible for its execution.

VI. A CASE STUDY: THE FIRST PROVENANCE CHALLENGE

As a case study we present a workflow implementing the First Provenance Challenge [21].

The goal of the First Provenance Challenge (FPC) workflow is to create a *brain atlas* from an archive of four high

resolution anatomical data and a reference image. Some image processing services are required for the workflow execution. These services have been deployed into heterogenous grid middlewares (more specifically, into the Condor cluster hosted by the I3A Institute and the gLite grids hosted by the BIFI Institute). In this example we show the flexibility of our proposal: some jobs are programmed to be executed by a specific computing platform, and other jobs may be executed by any available computing platform able to invoke the required service.

The workflow requires seven input parameters, whose specific values are implemented as the initial markings of places `Grid_Environment`, `Reference_image`, `Input_image_{1..4}`, and `Images_directory`. Their meanings are, respectively: the URL of one of the clusters where the workflow is going to be executed (more specifically, the cluster hosted by the I3A), the URI of the reference image, the URIs of the four images to be processed and the directory where the intermediate and final image files will be stored.

Figure 4 shows the implementation of the workflow using the Renew tool. Due to space limitations, only the first image processing flow is detailed in the figure, although the remaining branches for anatomy `Image2`, `Image3` and `Image4` are similar. Alternatively, Figure 5 depicts the implementation of the same workflow using Taverna. Job requests and results are encoded as Linda tuples. A request tuple is a nested tuple composed of four elements: the application or service to be executed and the URIs of the input and output data, the file descriptors for standard streams, QoS parameters and the computing platform where the request is going to be executed, respectively. Let us explain a tuple example, specifically the tuple depicted in transition `Align_warp_1(out)`. By putting that tuple in the message repository, the `Align_warp` service is invoked by the corresponding mediator using as input data an anatomy image, a reference image and their headers. The output is a warped image. For the sake of simplicity, file descriptors and QoS parameters are omitted in the tuple. Finally, the initial marking of the `grid_environment` place determines the value of the `grid` variable and, therefore, the computing platform selected for the job execution (the first field of this last tuple contains the access information required by the platform).

Tuples are either built and put into the message repository by means of the `Broker.out` action (as in the `Align_warp_1 (out) transition`, for instance) or withdrawn from the broker by means of the `Broker.in` action (as in the `Align_warp_1 (in) transition`, for instance). The sequential execution of these couple of transitions for a given image corresponds to an asynchronous call to the `Align_warp` service: first, the tuple with the information is put into the message broker, then the corresponding mediator takes it and invokes the service, putting the invocation result into the broker as a tuple and finally the result is captured and put into the workflow net by means of the second transition. Given the semantics of Petri nets, the processing of the input images can be done in any interleaved

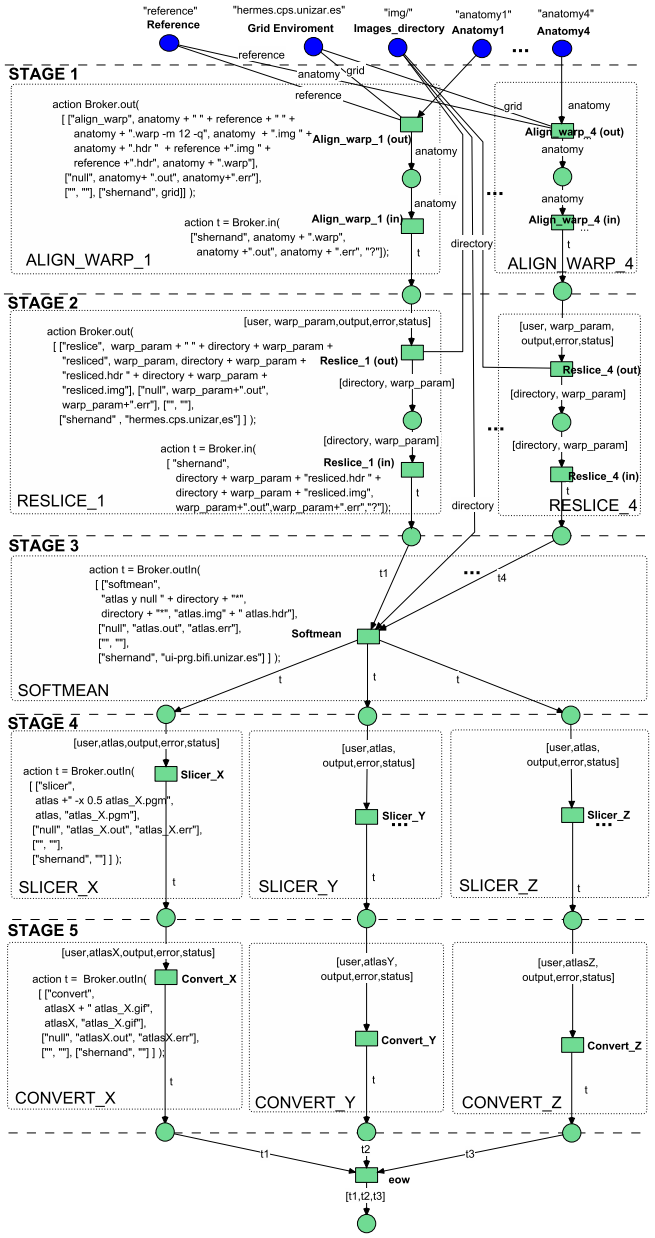

Fig. 4. Nets-within-Nets based implementation of the First Provenance Challenge workflow.

way, since tuples are put/removed into/from the broker as soon as resources are available. In this first stage the job request is executed in the cluster specified by the initial marking (the `grid` variable is an input parameter of the request submitted to the broker by the `Align_warp (out)` transition).

Once stages 1 and 2 are finished, Stage 3 takes the whole set of images from the directory specified by the parameter `Images_directory`, and executes the `softmean` method with these images as an input. At this stage the service deployed in one of the grids hosted by the BIFI institute is explicitly invoked. The last job request and its result are carried out by means of the `Broker.outIn` action: from the workflow point of view this corresponds to a synchronous call

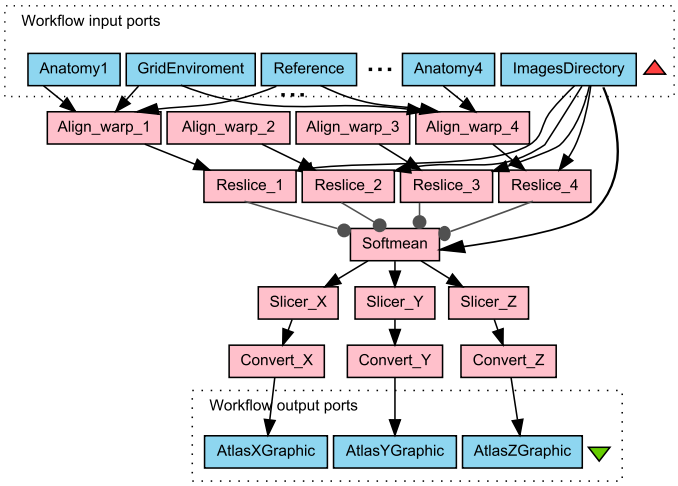

Fig. 5. Taverna implementation of the First Provenance Challenge workflow.

to the service described in the tuple. Then, softmean results are distributed so that stages 4 and 5 could be executed in parallel to compute the atlas data set for each dimension in axis x, y and z. The slicer and convert jobs could be executed by any available computing platform. Therefore, different executions of the workflow could invoke services deployed in different platforms. Finally, firing of transition `eow` (*end-of-workflow*) terminates the workflow. The resulting images will have been stored in the images directory.

Figure 5 depicts the workflow implemented with Taverna (some flow symbols in the top of the figure have been removed to improve readability). As shown, the structure is similar to the Nets-within-Nets implementation, although in this case the workflow is composed of several subworkflows, each of them implementing the previous invocations to the broker in order to put and withdraw tuples. Due to space limitations, the description of these subworkflows is left out of this paper.

A. Flexible deployment and execution

In order to analyze and test the transparency and flexibility of the proposed approach, the First Provenance Challenge workflow was executed using the framework. The target computing infrastructure for the execution of each stage (which can be specified in out transitions at each stage in Figure 5) was left unset, meaning that the mediators compete for each submitted task. At this respect, both HERMES and AraGrid were setup to separately allow the execution of the FPC workflow. However, as the aim of this experiment was to improve the overall execution cost of the workflow, the advanced scheduling component was programmed to perform a meta-scheduling process considering the load of the underlying computing infrastructures and the history of previous executions. Therefore, at every moment the best suitable candidate is estimated, avoiding the dispatching of a task to an overloaded infrastructure. This means that each task is first captured by the advanced scheduling component and then the target infrastructure is set (so the corresponding mediator will retrieve the task for its execution). However, the whole process is transparent from the user's perspective.

To do that, the advanced scheduler also considered the average load of each infrastructure at every moment. Figure 6 depicts the daily average load (% of the maximum load) in the HERMES and AraGrid computing infrastructures. As it can be observed, both computing infrastructures have different load models. Their trends during the day as well as the previous execution time are used to decide the most suitable candidate for each task deployment.

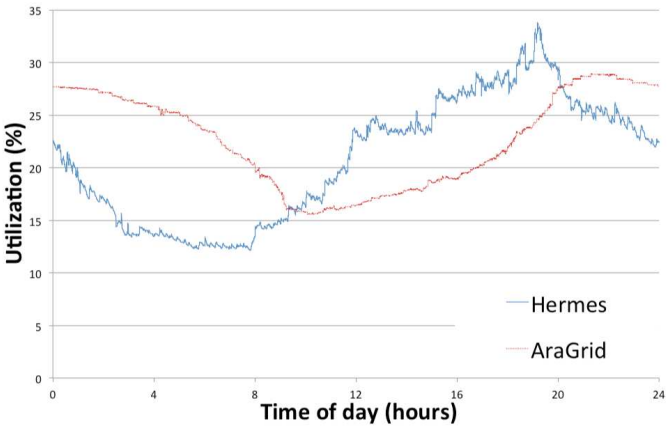

Fig. 6. Hermes and AraGrid daily utilization (in percentaje).

Figure 7 depicts the results obtained for 900 executions of the FPC workflow deployed on the framework. Average execution times (in seconds) are shown for each separated infrastructure (HERMES and AraGrid) and also for the framework for each stage of the First Provenance Challenge workflow. The overall execution time (average) is better when using the framework. This is due to the best candidate selection performed by the advanced scheduler (in most cases). The analysis of each separated stage depics that most of the time (70%) the HERMES cluster computing infrastructure gets a better execution time that AraGrid, which is related to the fact that the framework execution time is closed to the HERMES one.

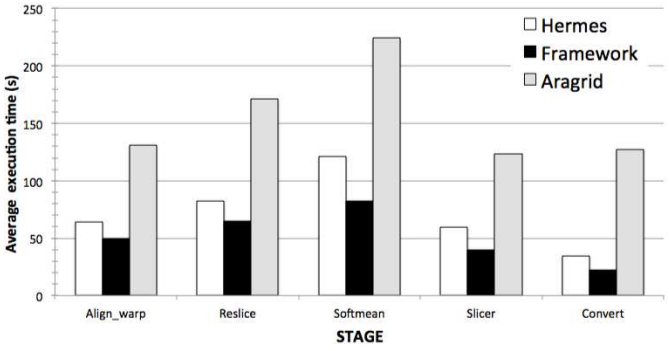

Fig. 7. Experimental results for the First Provenance Challenge workflow.

If we consider the average execution times for the complete workflow, AraGrid got the worst results with 777 seconds, HERMES got 362 seconds and the framework got 260 seconds. Obviously, using the most adequate infrastructure to get

the better execution time is not a trivial process from the researcher's point of view. However, by means of the use of the framework, this is done in a flexible and transparent way. Other possibilities are to reduce access costs (for instance, if each computing hour has an asssociated cost), resource usage, etc. Regarding the time to move data between the two infrastructures (as output from a stage is used as input of the following one), the average time for each workflow execution was less than 55 seconds (so the average framework execution time goes to 315 seconds).

VII. Conclusions

In this paper, a framework to solve some of the open challenges in the application of grid-based solutions to scientific workflows has been presented. This framework is uncoupled from specific grid technologies, able to work simultaneously and transparently with different and heterogeneous grid middlewares, providing scientists with a high level of abstraction when developing their workflows. The integration of the execution environment with different grid middlewares has been carried out by means of a resource broker composed of a Linda-based coordination system and a set of mediators. Thanks to the aforementioned broker, this integration is flexible and scalable. On the other hand, regarding the workflow programming point of view, the proposal is also open and flexible. As it has been shown, workflows programmed using standard languages or existing service-oriented workflow management systems (e.g. Taverna) can also be executed in the framework.

Currently, the proposed framework is being applied to solve some complex and high time-consuming problems, such as the behavioural analysis of semantically-annotated scientific workflows, or the analysis of existing data connections into the Linked data cloud, for instance. These solutions will allow improving the capabilities of the presented approach and also analyzing its performances.

We are also working on the integration of Cloud-related solutions, such as using the *Amazon Elastic Cloud Computing Simple Queue Service* (Amazon EC2 SQS) in order to have an alternative message repository, as well as providing specific high-performance computing capabilities (indeed, currently Amazon EC2 offers a mechanism to virtualize a HPC machine, able to handle critical and complex computation tasks). Related to this last point, we are adding some external reliable computing platforms by means of virtualization technologies. In [2] we sketched the implementation of a similar mediator able to support the execution of business tasks. Similarly, a new mediator able to submit job requests to the EC2 interface with the required policies has been implemented.

Acknowledgment

This work has been supported by the research project TIN2010-17905, granted by the Spanish Ministry of Science and Innovation.

References

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Second edition, Morgan Kaufmann Publishers, 2004.

[2] J. Fabra, P. Álvarez, J.A. Bañares, and J. Ezpeleta, *DENEB: A Platform for the Development and Execution of Interoperable Dynamic Web Processes*. Concurrency and Computation: Practice and Experience, Vol. 23, Issue 18, pp. 2421-2451, 2011.

[3] R. Tolosana-Calasanz, J.A. Bañares, P. Álvarez, and J. Ezpeleta. *Vega: A Service-Oriented Grid Workflow Management System*. In 2nd International Conference on Grid computing, High Performance and Distributed Aplications (GADA'07), vol. 4805, pp. 1516-1523, 2007.

[4] N. Carriero and D. Gelernter, *Linda in context*. Communications of the ACM, Vol. 32, Num. 4, pp. 444-458, 1989.

[5] O. Kummer, *Introduction to petri nets and reference nets*. Sozionik Aktuell, Num. 1, pp. 19, 2001.

[6] O. Kummer and F. Wienberg, *Renew - the Reference Net Workshop. Tool Demonstrations*. In 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), pp. 87-89, 2000.

[7] M. Rahman, R. Ranjan, and R. Buyya, *A Taxonomy of Autonomic Application Management in Grids*. In 16th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010), 2010.

[8] J. Yu and R. Buyya, *A taxonomy of workflow management systems for Grid Computing*. Journal of Grid Computing, Vol. 3, Issues 3-4, pp. 171-200, 2005.

[9] E. Huedo, R.S. Montero, and I.M. Llorente, *A Framework for Adaptive Execution on Grids*. Software - Practice and Experience , Vol. 34, Issue 7, pp. 631-651, 2004.

[10] M. Heidt, T. Dornemann, J. Dornemann, and B. Freisleben, *Omnivore: Integration of Grid meta-scheduling and peer-to-peer technologies*. In 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), pp. 316-323, 2008.

[11] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, and I.M. Llorente, *Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers*. Future Generation Computer Systems, Vol. 28, pp. 358-367, 2012.

[12] J. Yu and R. Buyya, *A novel architecture for realizing Grid Workflows using Tuple Spaces*. In 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004), Pittsburgh, USA, pp. 119-128, 2004.

[13] G. Mateescu, W. Gentzsch, and C.J. Ribbens, *Hybrid computing: Where HPC meets grid and Cloud computing*. Future Generation Computer Systems, Vol. 27, pp. 440-453, 2011.

[14] Y. Zhang, C. Koelbel, and K. Cooper, *Hybrid re-scheduling mechanisms for workflow applications on multi-cluster grid*. In 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), pp. 116-123, Shanghai (China), 2009.

[15] *Taverna: an open source and domain independent Workflow Management System*. Available at http://www.taverna.org.uk/, 2011.

[16] P. Álvarez, J. A. Bañares, and P.R. Muro-Medrano, *An Architectural Pattern to Extend the Interaction Model between Web-Services: The Location-Based Service Context*. In 1st International Conference on Service Oriented Computing (ICSOC 2003), Lecture Notes in Computer Science, Vol. 2910, pp. 271-286, 2003.

[17] J. Fabra, P. Álvarez, and J. Ezpeleta, *DRLinda: A distributed message broker for collaborative interactions among business processes*. In 8th International Conference on Electronic Commerce and Web Technologies (EC-Web 2007), Lecture Notes in Computer Science, Vol. 4655, pp. 212-221, 2007.

[18] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, *Job Submission Description Language (JSDL) Specification, Version 1.0*. Available at http://www.gridforum.org/documents/GFD.56.pdf, 2011

[19] R. Sharma, V.K. Soni, M.K. Mishra, and P. Bhuyan, *A survey of job scheduling and resource management in grid computing*. World Academy of Science, Engineering and Technology, Issue 64, pp. 461-466, 2010.

[20] H. Boley, *Rule Markup Language, RuleML Specification. Version 1.0.*. Available at http://ruleml.org/, 2011.

[21] L. Moreau et al., *The First Provenance Challenge*. Concurrency and Computation: Practice and Experience, Vol. 20, Issue 5, pp. 409-418, 2008.

Anexo B | *A Simulation-Based Scheduling Strategy for Scientific Workflows*

En este Anexo se muestra el artículo titulado "*A Simulation-Based Scheduling Strategy for Scientific Workflows*" el cual ha sido aceptado para su publicación en la *II International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH 2012) que se celebrará en Roma (Italia), del 28 al 31 de Julio de 2012.

El artículo se presenta manteniendo el formato con el que aparecerá en el congreso mencionado.

A SIMULATION-BASED SCHEDULING STRATEGY FOR SCIENTIFIC WORKFLOWS

Sergio Hernández, Javier Fabra, Pedro Álvarez, Joaquín Ezpeleta
Aragón Institute of Engineering Research (I3A)
Department of Computer Science and Systems Engineering
University of Zaragoza, Spain
{*shernandez, jfabra, alvaper, ezpeleta*}*@unizar.es*

Keywords: Scientific workflows, Grid modelling and simulation, Workloads, Performance analysis.

Abstract: Grid computing infrastructures have recently come up as computing environments able to manage heterogeneous and geographically distributed resources, being very suitable for the deployment and execution of scientific workflows. An emerging topic in this discipline is the improvement of the scheduling process and the overall execution requirements by means of simulation environments. In this work, a simulation component based on realistic workload usage is presented and integrated into a framework for the flexible deployment of scientific workflows in Grid environments. This framework allows researchers to simultaneously work with different and heterogeneous Grid middlewares in a transparent way and also provides a high level of abstraction when developing their workflows. The approach presented here allows to model and simulate different computing infrastructures, helping in the scheduling process and improving the deployment and execution requirements in terms of performance, resource usage, cost, etc. As a use case, the Inspiral analysis workflow is executed on two different computing infrastructures, reducing the overall execution cost.

1 INTRODUCTION

Grid computing emerged as a paradigm for the development of computing infrastructures able to share heterogeneous and geographically distributed resources (Foster and Kesselman, 2003). Due to their computational and networking capabilities, this type of infrastructure has turned into execution environments suitable for scientific workflows, which require intensive computations as well as complex data management. Nevertheless, the comparison of existing Grid workflow systems has shown relevant differences in the building and execution of workflows that causes experiments programmed by scientists and engineers to be strongly coupled to the underlying system responsible for their execution (Rahman et al., 2011; Yu and Buyya, 2005). Therefore, two of the most interesting open challenges in the field of scientific computing are the ability to program scientific workflows independently of the execution environment and the flexible integration of heterogeneous execution environments to create more powerful computing infrastructures for their execution.

This new generation of computing infrastructures requires new strategies of resource brokering and scheduling to facilitate the utilization of multiple-domain resources and the allocation and binding of workflow activities to them. An emerging topic in this discipline is the use of simulation environments to help in the scheduling process, improving the overall execution requirements in terms of resource usage, time and costs. Some approaches such as GMBS (Kertész and Kacsuk, 2010) or SCI-BUS [1], for instance, propose the use of simulation tools to evaluate the best meta-scheduling strategy. Different scheduling policies can be evaluated to decide the most suitable allocation of workflow activities to resources. On the other hand, another research focus on the development of a novel scheduling algorithm and its execution over a simulated environment. The results are then compared with other similar algorithms in order to classify the algorithm with respect to some predefined criteria. Strategies are normally compared in terms of makespan (Hamscher et al., 2000; Abraham et al., 2006; Yu and Shi, 2007), simulation times (Ludwig and Moallem, 2011) or queue times (Yu and Shi, 2007; Ludwig and Moallem, 2011).

[1] http://www.sci-bus.eu/

Regardless of the problem to be solved, simulation environments may consider execution environment models and workloads with the purpose of improving scheduling decisions. The first provide a complete specification of architectures and configurations of the execution environment. Flexible mechanisms for the specification of these models should be provided, specially to model evolving and heterogeneous computing infrastructures. Meanwhile, workloads are logs of job sets based on historical data or statistical models representing jobs to be executed in the environment. The relation between workloads and scheduling policies turns around the necessity of using a workload fitting the characteristics of jobs executed in the infrastructure in order to evaluate the suitability of a concrete scheduling algorithm in real terms. In (Feitelson, 2002), the benefits of using workloads as well as how to use them to evaluate a system are discussed. However, their use is still rather limited, due mainly to the complexity of its creation, being the process automation a difficult task. Therefore, workloads are mainly used just for the analysis of Grid systems (Iosup and Epema, 2011; Li et al., 2004). Understanding these real workloads is a must for the tuning of existing Grids and also for the design of future Grids and Cloud infrastructures.

In (Fabra et al., 2012), a framework for the deployment and execution of scientific workflows whose main features are described in Section 2 was presented. This framework facilitates the flexible integration of heterogeneous Grid computing environments, addressing the challenge of creating more powerful infrastructures. Besides, its architectural design guarantees that workflow programmers do not need to be aware of this heterogeneity. In this paper, we integrate new components into our framework for the simulation of scientific workflows using realistic workloads, allowing the improvement and flexibility of job allocation by means of a meta-scheduler. Unlike other approaches which are focused on assisting the researcher, in our proposal simulation results are internally used to make scheduling decisions transparently to researchers and their workflows. Obviously, the complexity of this simulation-based scheduling is increased by the evolving nature of the underlying computing infrastructure.

The information obtained from the simulator component can also be used by the meta-scheduler in order to carry out some optimization process depending on the parameters to be optimized. For instance, it is possible to provide a better-execution-time algorithm which schedules the execution of jobs on the most suitable computing infrastructure depending on the workload provided at the execution time. It is also possible to easily minimize resource costs while keeping a defined relation between execution time and involved costs, for instance.

The remainder of this paper is organized as follows. The main features of the developed framework in which the presented simulation approach is integrated are described in Section 2. The design and implementation of the simulator is sketched by means of the application to a real cluster which uses Condor in Section 3. The flexibility and reuse capabilities of the component are then depicted in Section 4 by means of the integration of another real Grid managed by gLite. Then, the simulation approach integration is applied to the development of a real case study, the LIGO Inspiral analysis workflow in Section 5. Finally, Section 6 concludes the paper and addresses future research directions.

2 EVOLVING TOWARDS THE ADAPTABLE DEPLOYMENT OF SCIENTIFIC WORKFLOWS

The proposed Grid-based framework for programming and executing scientific workflows is able to tackle some of the open challenges in the field of Grid computing. From the programmer's point of view, workflows can be programmed independently of the execution environment where the related tasks will be executed. Different standard languages, widely accepted by the scientific community (e.g. Taverna), can be used for programming this type of abstract workflows. On the other hand, the proposed framework is open and flexible from the computing resource integration's point of view. First, and in accordance with this feature, it is able to simultaneously work with different Grid middlewares or middlewares implemented using other alternatives (e.g. Web services). And, secondly, heterogeneous execution environments can be added, modified or even removed without previous announcement and in a flexible and transparent way. Therefore, the combination of these features turns our solution into a novel and suitable proposal in the field of scientific workflows (Yu and Buyya, 2005).

Figure 1 shows the high-level architecture of the proposed framework. A more detailed description is outside the scope of this paper. Let us concentrate on the process of executing workflow tasks and the architectural components involved in it.

Once a workflow has been deployed, the *workflow execution environment* is responsible for controlling its execution and submitting tasks to the *resource*

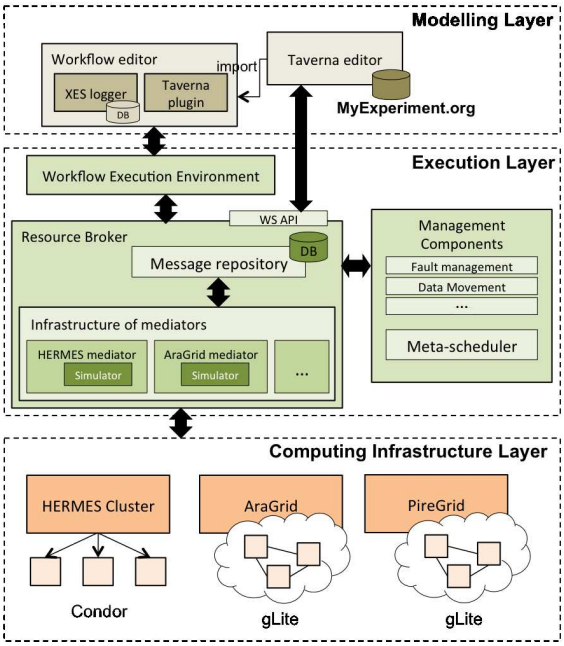

Figure 1: Architecture of the execution environment.

broker by means of its interface as they must be executed. Submitted tasks are then stored into the *message repository* as messages that encapsulate the information needed for the execution of a task, including the application to be executed, the references to input and output data, a description of the resources required for its execution (operating system, CPU architecture and features, memory, network bandwidth, etc.) and QoS parameters. These messages are described using the JSDL standard. Optionally, the target computing environment responsible for the task execution can be also included into the message. This type of tasks is called *concrete tasks*. Nevertheless, workflows will be usually programmed independently of the execution environment where their tasks will be executed (*abstract tasks*). This decision tries to take full advantage of the integration capabilities of rid-based framework.

An *infrastructure of mediators* uncouples the resource broker from the specific and technological details about the Grid-based computing environments where tasks will be executed. Each computing environment is represented by a mediator. Internally, a mediator handles a complete information about the Grid infrastructure it represents. Subsequently, this knowledge will be used by the mediator to interact with the message repository and to find at run-time abstract tasks that could be executed by its middleware. Therefore, mediators are responsible for making dispatching decisions related to the execution of tasks. Obviously, in this dispatching model more than one mediator could compete for the execution of a specific task (the criterion would be that their corresponding middlewares were able to execute it). This proposal is an alternative to traditional solutions based on the use of a centralized task scheduler responsible for deciding where tasks will be executed.

Finally, each mediator dispatches its tasks to the middleware managing the infrastructure it represents for their execution and stores the results of the executed tasks into the message repository, as well as the resulting execution log, which can be used for monitoring or analysis purposes. These results will be subsequently recovered by the workflow execution environment for controlling the execution of the deployed workflow.

2.1 Improving the scheduling capabilities of the framework

The dispatching strategy of our proposal presents a set of drawbacks: 1) performance issues related to the execution of tasks are not considered by mediators (therefore, a task could be executed by an inappropriate computing environment degrading the performance of the whole workflow); 2) dispatching decisions are locally adopted by each mediator and, consequently, one of them could monopolize the execution of pending tasks (this could cause unnecessary overloads on its corresponding computing environment); and, finally, 3) the real behaviour of the existing computing environments and the state of their resources is also ignored by the mediators.

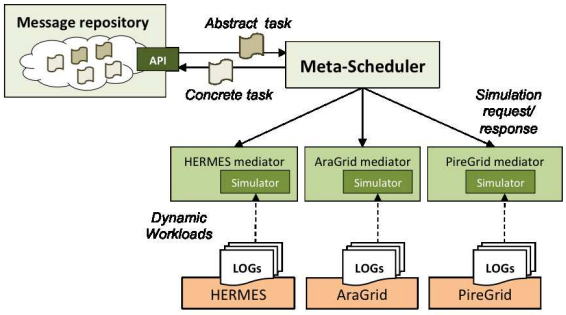

Figure 2: Architectural components for the simulation-based scheduling.

In order to solve the previous drawbacks and also to improve and enhance our infrastructure, a *meta-scheduler* based on simulation techniques will be integrated into the Grid-based framework in this paper. Figure 2 represents the alternative process of executing workflow tasks using a meta-scheduler. Initially, pending (abstract) tasks are stored into the message repository. The meta-scheduler retrieves this type of tasks for determining where they will be finally executed. Scheduling decisions are made by simulat-

ing the execution of each task in the existing computing environments and analysing the simulation results. With these results, the task is made concrete and then submitted to the message repository, allowing the task to be executed by the selected mediator.

The interface of mediators has been extended to support this process. Now, each mediator exposes a set of operations able to simulate the execution of a task. Internally, a simulator has been integrated into each mediator for providing the required functionality. More specifically, the simulator is able to: 1) model the corresponding computing environment managed by the mediator (computing resources, memory, network bandwidth, user and scheduling internal policies, etc.); 2) select the most suitable workload for representing the real behaviour of the computing environment and the state of its resources (execution logs are used for creating these workloads); and, finally, 3) simulate the execution of tasks measuring parameters such as the execution time, the data transfer time, the queuing time, the consumed memory, etc.

In the following, the design and implementation of the simulator component is depicted. As it will be shown, this component is flexible enough as to allow an easy adaptation for different computing infrastructures with different scheduling policies.

3 SIMULATING WORKFLOW'S EXECUTION

As stated, the simulator component has been integrated as an internal component in each mediator. Therefore, each computing infrastructure can handle different and customized simulation capabilities. Anyway, simulators are accessed through a well defined API, so adding new simulators to the framework is a guided and easy process. Also, coupling simulation components with mediators allows developers to introduce new computing infrastructures without needing to implement them. Obly the corresponding scheduling policy and the associated simulator must be considered.

The simulation component receives the Grid model and the workload as an input, which are stored as files accessible from the corresponding mediator. Then, after a processing cycle, it generates as a result the execution estimation in terms of time and resource usage with respect to the input provided. The simulator also provides some metrics for analysis purposes such as the average system utilization of each resource, for instance, which can be used to improve the process.

In the following, the design and implementation of the simulation component is sketched by means of the description of two real use cases: the HERMES cluster and the AraGrid multi-cluster Grid.

3.1 Overview of the HERMES cluster

HERMES is a cluster hosted by the Aragón Institute of Engineering Research (I3A)[2]. In general terms, HERMES consists of 1308 cores and 2.56 TB of RAM. More specifically, it consists of 126 heterogeneous computing nodes, including 52 nodes with two 2.33 GHz 4-core Intel Nehalem CPUs and 24 GB of RAM per node, 48 nodes with two 2.00 GHz 8-core AMD Magny-Cours CPUs and 16 GB of RAM per node, 12 nodes with a 3.00 GHz 4-core Intel Woodcreest quadcore CPUs and 8 GB of RAM per node, 11 nodes with two 2.33 GHz 2-core Intel Woodcreest CPUs and 4 GB of RAM per node, and 4 nodes with two 2.66 GHz 4-core Intel Woodcrest CPUs and 16 GB of RAM per node. The computing nodes in HERMES are connected by Gigabit links, allowing high-speed data transfers.

At the moment of this writing, the cluster is managed by the Condor[3] middleware version 7.6.3.

The cluster is used by a vast variety of researchers, mainly focused on inductive and physical systems, automotive systems, discrete event system analysis and complex semantic workflow analysis. System utilization is usually focused on the use of CPUs rather than memory consumption. Data inputs are usually small sized, although there is a group handling complex experiments with files of more than 20TB. The analysis of relevant workloads shown that the average user is not aware of load peaks or advanced configuration issues, which normally produces that experiments last extremely long, require oversized resources or even are queued for long times. In this scenario, our proposal for a framework which would optimize such situations is extremely useful from both the researcher and also the system usage perspectives.

3.2 Implementation details of the HERMES simulator

Alea (Klusáček and Rudová, 2010) has been used to implement the internal simulator in the HERMES mediator component. Alea is an event-based simulator built upon the GridSim toolkit (Sulistio et al., 2008). Alea extends GridSim and provides a central scheduler, extending some functionalities and improving

[2] http://i3a.unizar.es

[3] http://research.cs.wisc.edu/condor/

scalability and simulation speed. Alea has been designed to allow an easy incorporation of new scheduling policies and to easily extend its functionalities. Also, Alea provides an experimentation environment easy to configure and use, which helps in the quick and rapid development of simulators when a new infrastructure is going to be added to the system.

The original implementation of Alea has been extended to allow some Condor features such as user priorities, RAM requirements and preemptions. Figure 3 depicts the structure of the simulator. As shown, it consists of two input files, the *workload* and the *Grid model*, and four main modules, the *Job Loader*, the *Machine Loader*, the *Scheduler* and the *Result Collector*, respectively.

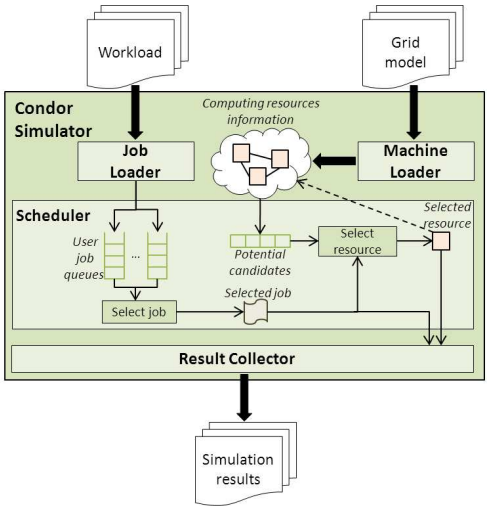

Figure 3: Architecture of the Condor simulator based on Alea.

Multiple workload have been composed using the cluster execution logs from the last year and identifying common situations of resource utilization and job submission. The workload is represented using the Grid Workload Format (GWF format) proposed by the Grid Workload Archive (GWA) (Iosup et al., 2008a). For each job, the job execution time, the number of CPUs required, the memory requirements, the user and group who executes the job and the job dependencies (if exists) are provided. More details on the creation of workloads is provided in subsection 5.1.

The *Grid model* is a text file that contains the information of each computing node. The representation of each node includes a node identifier, the number of machines, the number of CPUs per machine, the total amount of memory per machine, the system architecture, its operating system and the network characteristics. Also, a failure model can be detailed to reflect dynamic changes in the infrastructure during the simulation.

The *Job Loader* component reads the job descriptions and sends them to the scheduler. This module has been extended to allow RAM requirements and user and group details of the submitted jobs.

The *Machine Loader* component is responsible for reading the resource description from a file containing the Grid model. This module has been extended to be able to parse and save the information provided.

The *Scheduler* component is the more complex one. It has been extended with a new scheduling policy considering the schema for user priorities that Condor applies in HERMES. This scheduling policy works as follows: when a job sent by the Job Loader reaches the scheduler, the job is queued in the right user queue. This queue is ordered by the job priority and the job arrival time. When the scheduler requests a new job to be executed, jobs are ordered by their user priority and the job with the highest priority is chosen. Then, the machines with available resources (CPUs and RAM) and also the machines that could have available resources (if some running jobs are evicted) are selected as potential candidates to execute the job. The list of all potential candidates is ordered by multiple criteria (job preferences, machine preferences, etc.) to get the most suitable resource. If there is no resource available to execute the job, this is queued again and the scheduler looks for the next job. Finally, when a job and a resource have been chosen, the job is sent to the resource and its state is updated. In addition, some of the current running jobs are evicted from the selected resource if necessary to execute the new job. These evicted jobs are requeued and will be reexecuted later.

Finally, the Result Collector component is responsible for storing the simulation results and provide them as output. When a job is sent to a resource, evicted or a machine fails, the Result Collector stores this information. When a job ends, the Result Collector stores the job information in an output file. For each job, the arrival time, the time the job has spent queued, the execution time of the resource, the resource where the job was executed and the number of evictions suffered by the job are stored in the file.

3.3 Validation of the HERMES simulator

The aim of the developed simulator is to be used as a decision tool at meta-scheduling level. In terms of simulation accuracy, its validation is a key issue to verify its feasibility and usefulness for this purpose (Sargent, 2010). Figure 4 shows a comparison of the actual cluster utilization, extracted from the logs, and

the simulated utilization, obtained from the simulation of the tasks described in the workload. The comparison is presented as a daily cycle in which the horizontal axis indicates the time (in hours) and the vertical axis shows the CPU utilization rate (in percentage). As it can be observed, the simulation results are very similar to real results. Both plots follow the same trend, being the simulation utilization slightly lower. In terms of the deviation of the simulation results, an average error of 15.09% and a standard deviation of 8.03% is observed.

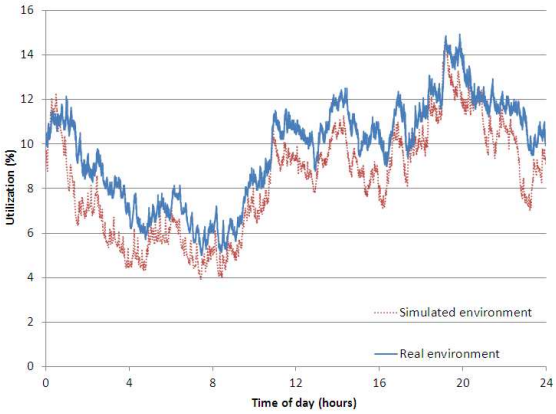

Figure 4: Condor cluster utilization for the real and simulated environment.

In order to validate the job performance indicator, two metrics are provided: the execution time and the queue time. Figure 5 shows the cumulative distribution function for the execution time (Figure 5-a) and the queue time (Figure 5-b). For the sake of clarity, the horizontal axis is shown on a log scale. Figure 5-a illustrates that job execution time is almost the same in the simulation and the real environment. In contrast, there is an important difference between queue time in both environments, which can be explained because the simulator is able to schedule a job without delay when there are available resources to execute a job. However, Condor middleware suffers for several delays due to different reasons such as delay notifications between distributed components, scheduling cycle duration or status update. To fix this error and reduce its influence on the results, two techniques are proposed: the first one adds a synthetic delay to the job execution time, whereas the second one adds the synthetic delay to the job queue time results. Also, how this feature can be incorporated in the simulator to get more accurate simulations is being studied for the meantime.

4 EXPERIENCE REUSE FOR THE SIMULATION OF A GLITE GRID

In this section, how a simulator for a multi-cluster Grid can be easily implemented replacing some parts of the previously developed simulator is shown. Also, we illustrate the usefulness of the methodology presented to validate the simulator results.

4.1 Overview of the AraGrid Grid

AraGrid[4] is a research and production Grid hosted by the Institute for Biocomputation and Physics of Complex Systems (BIFI)[5] and it is part of the European Grid Initiative (EGI)[6]. AraGrid consists of four homogeneous sites located at four different faculties in different geolocated cities. Every site is formed by 36 nodes with two 2.67 GHz 6-core Intel Xeon X5650 CPUs and 24 GB of RAM per node, making a total amount of 1728 cores and 4 TB of RAM. Both sites and nodes are interconnected by Gigabit links.

The Grid is managed by the gLite[7] middleware version 3.2.0 and every site use openPBS[8] version 2.5 as local batch system.

The AraGrid infrastructure is oriented to long-term experiment in the fields of physics, biochemistry, social behaviour analysis, astronomy, etc. Users are more conscious of loads and resource usage, although they deploy experiments similarly to the HERMES case, getting long waiting times.

4.2 Implementation and validation of the AraGrid simulator

Starting from the simulator structure, the design and implementation of the Condor simulator has been reused to develop a gLite simulator valid for the AraGrid computing infrastructure. This is an easy and quick implementation process, and the resulting simulator can be easily adapted to another gLite infrastructure. The reasons to implement these two simulators is twofold. On the one hand, HERMES (managed using Condor), AraGrid (gLite) and also PireGrid (gLite) are connected using high speed Gigabit links, which enhances data movement performance (which is left out of the scope of this paper). On the other hand, Condor and gLite are well known and

[4] http://www.araGrid.es/
[5] http://bifi.es/es/
[6] http://www.egi.eu/
[7] http://glite.cern.ch/
[8] http://www.mcs.anl.gov/research/projects/openpbs/

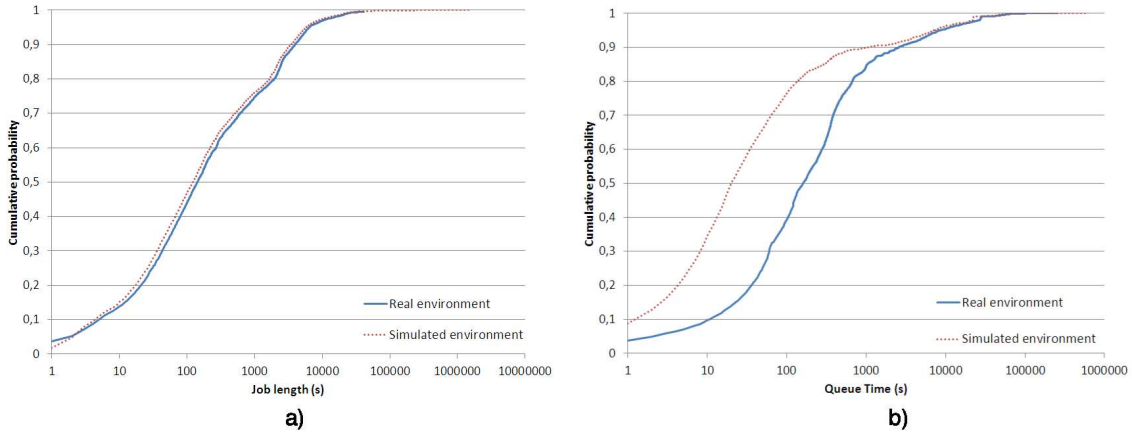

Figure 5: Job performance comparison between real data and simulation results in terms of: (a) job execution time, (b) job queue time.

widespread cluster/Grid middlewares in the research community.

The only component that needed a custom adaptation to fit the behaviour of AraGrid with respect to the HERMES simulator component is the scheduler. The scheduler's policy follows a hierarchical approach, as shown in Figure 6. Jobs sent by the Job Loader are managed by the global scheduler component that sends them to the right local scheduler considering job requirements, job rank and site occupation are taken. Meanwhile, every local scheduler uses a custom First Come First Serve (FCFS) policy.

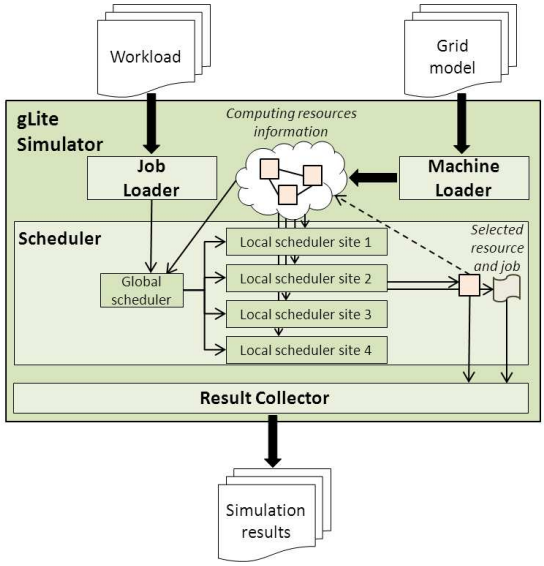

Figure 6: Architecture of the gLite simulator and detail of the local scheduler of a site.

It is important to consider a special case. As some sites are shared with other Grid initiatives such as EGI, the workload used as input contains jobs that can only be executed in shared sites. Sites where a job can be executed depends on the Virtual Organization (VO). Since this information is included in the workload, this special case can be properly treated by the scheduler when this kind of job reaches the global scheduler.

The resulting simulator component has been integrated into the AraGrid gLite mediator. The validation of the component has been carried out following the same approach depicted in subsection 3.3. In this case, the results are more accurate than in the HERMES case. That is because AraGrid scheduling policy is easier to replicate. The average error is of 1.19% with a standard deviation of 0.85%.

5 A CASE STUDY: INSPIRAL ANALYSIS WORKFLOW

In this section, the proposed simulation-based approach is applied in order to improve the performance of the Inspiral analysis scientific workflow. The experiment setup is detailed, with particular attention to the workload creation method used for modelling other users jobs that are executed in HERMES and AraGrid at the same time. Finally, performance results showing the benefits of our infrastructure are presented and discussed.

One of the main research lines of the Laser Interferometer Gravitational Wave Observatory (LIGO) is the detection of gravitational waves produced by various events in the universe (based on Einstein's theory of general relativity). The LIGO Inspiral Analysis Workflow is a scientific workflow which analyzes and tries to detect gravitational waves produced by various events in the universe using data obtained from the coalescing of compact binary systems such

as binary neutron stars and black holes (Taylor et al., 2006). Figure 7 depicts a simplified view of the main structure of the workflow. Although the workflow has a simple structure, it allows a high level of parallelism. As shown, the whole experiment is split into several smaller stages or blocks for analysis. The time-frequency data from any event for each of the LIGO detectors is arranged into template banks and used as an input for the workflow, which generates a subset of waveforms belonging to the parameter space and computes the matched filter output in each stage. Inspiral jobs are the most computationally intensive tasks in the workflow, generating most of the computing requirements. In case a true inspiral is detected, the matched filter output is computed and a trigger is generated and tested for consistency by the Thinca jobs as a result from the experiment. Finally, template banks are then generated from these trigger outputs and the process repeats.

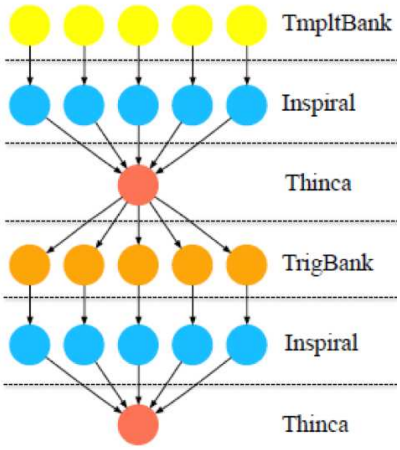

Figure 7: Workflow of the LIGO Inspiral analysis scientific workflow.

Several scientific workflows management systems could be used to develop the workflow. In our case, a high level Petri nets implementation (Reference nets) has been developed using the workflow editor provided by the framework depicted in Section 2. However, the workflow implementation details are out of the scope of this paper.

5.1 Experiment setup

The experiment setup is not specific for this experiment or case study, but it is a general setup automatically generated by the components of the framework. This design simplifies the use of the infrastructure, making the simulation-based meta-scheduling completely transparent to the user.

The process is as follows: first, when a mediator retrieves a simulation request, it builds a workload describing the tasks to be simulated. Next, it gets information about the state of the computing infrastructure it represents. These data are used to adapt the predefined Grid model to its current situation (introducing resource failures) and to build a second workload representing the infrastructure state during the simulation. Details about the creation of this second workload are shown below. Once both workloads have been created, they are combined into one that is used as the simulation input. Then, the simulation starts its execution. Once it has finished, the simulation results are analysed by the mediator and only the information concerning the target tasks is provided to the meta-scheduler. Finally, the meta-scheduler chooses the best computing infrastructure based on data obtained from several simulations. For that purpose, the meta-scheduling policy uses a better-execution-time algorithm. Nevertheless, more complex policies involving the information obtained in previous simulations could be easily used.

The creation of the workload used to represent the state of the computing infrastructure is a key step in the simulation process. The importance of using an appropriate workload has been identified as a crucial input in some previous work (Feitelson, 2002; Li et al., 2004). Using a wrong workload can cause the simulation results not to correspond to the actual behaviour of the involved Grids. These research papers propose the generation of a single workload based on historical information from a long period of time and only considering representative periods (e.g. the peak hours during weekdays in job-intensive months). It is assumed that the longer the observation period is, the more representative is the workload, which allows tunning the Grid in extreme situations (Feitelson, 2002). Nevertheless, for simulation purposes these approaches are not valid because the state of the resources must be considered as the simulation starts. If an average or extreme workload is used, it is very likely to get very inaccurate results that lead to wrong scheduling decisions. Our proposal is to build several representative workloads with different situations depending on the state of the infrastructure (e.g. low load, average load and high load) and date. Therefore, the current computing infrastructure state is obtained before starting a simulation and used to select the most suitable workload. Also, the recovered infrastructure information, including currently running jobs and queued jobs, is added at the beginning of the workload, obtaining this way a workload describing the current infrastructure state and its evolution.

The model proposed in (Iosup et al., 2008b) has been used for workload creation. This model incorpo-

rates the notions of different users and jobs in *Bag-of-Tasks* (BoTs) to the Lublin-Feitelson model (Lublin and Feitelson, 2003). Due to the fact that the HERMES and AraGrid analysis has shown that more than 90% of jobs belongs a BoT and a few users are responsible for the entire load, this model is suitable for modelling jobs in our infrastructures.

5.2 Analysis of the results

To prove the usefulness of the proposed approach, the workflow has been executed for a whole day (24 hours). Figure 8 depicts the CPU load observed in HERMES and AraGrid during the experiment. Note that HERMES load is different from the one sketched in figure 4. That is because the load in Figure 4 is an average load extracted from the execution log corresponding to the whole last year, whereas Figure 8 shows the cluster load on a particular day. As it can be observed, both computing infrastructures have different load models. Throughout the day there are better periods of time for submitting jobs to HERMES (mostly at early morning and night), and times more appropriate to submit jobs to AraGrid (in the afternoon). However, this is not the only criterion to be considered as the performance of a Grid infrastructure depends on many factors.

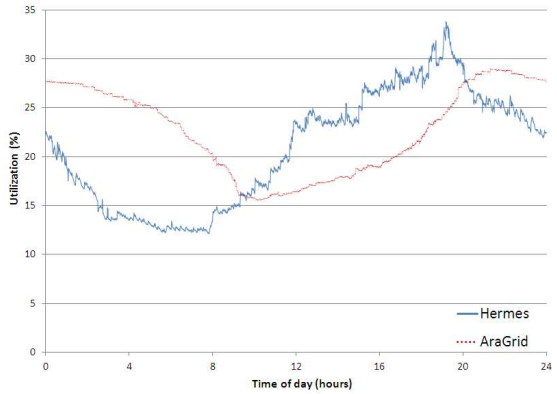

Figure 8: HERMES and AraGrid utilization (in percentage) observed during workflow execution.

The use of the simulation as a decision tool for meta-scheduling deals with this complexity and improves the performance obtained in the execution of the workflow as shown in Figure 9. The figure shows the total execution time for each stage of the Inspiral workflow entirely executed in each computing infrastructure (HERMES on the left bar and AraGrid on the right bar) and using the framework with the simulation-based meta-scheduling strategy (center bar) depicted previously. The results show that the use of the proposed approach leads to an improvement of 59% in HERMES execution time and a 111% in AraGrid execution time.

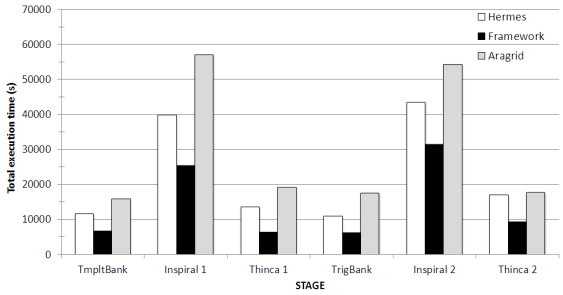

Figure 9: Experimental results for LIGO Inspiral analysis workflow.

Regarding the simulation overhead in terms of execution time, the simulation process for HERMES is more complex (more iterative structures) and can take up to 3-4 minutes for a bag of 10000 tasks, whereas for gLite it takes one minute approximately. Therefore, simulation times are insignificant in comparison to the execution time of each stage. Also, data movement has been measured. For the sake of clarity, as HERMES and AraGrid are connected by a Gigabit link, these times are small and can be avoided in the calculation of the overall execution time.

6 CONCLUSIONS

In this paper, a simulation component based on realistic workload usage has been presented. This component allows modelling and simulating different computing infrastructures in terms of performance, resource usage, cost, etc. We have also described a framework developed for the flexible deployment of scientific workflows in Grid environments, and which allows researchers to transparently work simultaneously with different and heterogeneous Grid middlewares.

The integration of the simulation component into the framework allows improving the meta-scheduling process. Not only a simulation process can be carried out to find the best computing infrastructure to execute a task (or a bag of tasks) in terms of performance or costs, but also the process may vary depending on the used workload. The use of realistic workloads provides very suitable and reliable results.

The flexible design and implementation of the simulation component also allows an easy adaptation for being used with different computing infrastructures, as it was shown by means of the reuse of the HERMES simulator component (Condor) to develop

the AraGrid one (gLite). Both Condor and gLite are two of the most used cluster/Grid middlewares in the research community. Thus, an additional advantage is that the developed components can be easily reused for simulating other existing computing infrastructures.

Finally, the integration of the presented approach into the framework has been applied to the development and execution of the Inspiral analysis over two different computing infrastructures, HERMES and AraGrid. As a result, the overall execution cost was significantly reduced.

Currently, the proposed simulation component is being extended to support the dynamic building of workloads. The use of dynamic workloads will minimize the effort required to build a new simulator and allow to obtain more accurate simulations. Also, the addition of new features in the simulator is being addressed in order to get more accurate queue times in simulations. Finally, the incorporation of complex meta-scheduling approaches that can use the information provided by the simulation process will be studied.

ACKNOWLEDGEMENTS

This work has been supported by the research project TIN2010-17905, granted by the Spanish Ministry of Science and Innovation.

REFERENCES

Abraham, A., Liu, H., Zhang, W., and Chang, T.-G. (2006). Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm. In *Proceedings of the 10th International Conference in Knowledge-Based Intelligent Information and Engineering Systems – KES 2006*, volume 4252, pages 500–507.

Fabra, J., Hernández, S., Álvarez, P., and Ezpeleta, J. (2012). A framework for the flexible deployment of scientific workflows in grid environments. In *Proceedings of the Third International Conference on Cloud Computing, GRIDs, and Virtualizations – CLOUD COMPUTING 2012*.

Feitelson, D. G. (2002). Workload Modeling for Performance Evaluation. In *Proceedings of Performance Evaluation of Complex Systems: Techniques and Tools – Performance 2002*, volume 2459, pages 114–141.

Foster, I. and Kesselman, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Hamscher, V., Schwiegelshohn, U., Streit, A., and Yahyapour, R. (2000). Evaluation of Job-Scheduling Strategies for Grid Computing. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing – GRID 2000*, pages 191–202.

Iosup, A. and Epema, D. H. J. (2011). Grid Computing Workloads. *IEEE Internet Computing*, 15(2):19–26.

Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., and Epema, D. H. J. (2008a). The Grid Workloads Archive. *Future Genereration Computer Systems*, 24(7):672–686.

Iosup, A., Sonmez, O., Anoep, S., and Epema, D. (2008b). The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th international symposium on High performance distributed computing – HPDC 2008*, pages 97–108.

Kertész, A. and Kacsuk, P. (2010). GMBS: A new middleware service for making grids interoperable. *Future Generation Computer Systems*, 26(4):542–553.

Klusáček, D. and Rudová, H. (2010). Alea 2 – Job Scheduling Simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques – SIMUTools 2010*.

Li, H., Groep, D., and Wolters, L. (2004). Workload characteristics of a multi-cluster supercomputer. In *Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing – JSSPP 2004*, pages 176–193.

Lublin, U. and Feitelson, D. G. (2003). The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122.

Ludwig, S. A. and Moallem, A. (2011). Swarm Intelligence Approaches for Grid Load Balancing. *Journal of Grid Computing*, 9(3):279–301.

Rahman, M., Ranjan, R., Buyya, R., and Benatallah, B. (2011). A taxonomy and survey on autonomic management of applications in grid computing environments. *Concurrency and Computation: Practice and Experience*, 23(16):1990–2019.

Sargent, R. G. (2010). Verification and validation of simulation models. In *Proceedings of the 2010 Winter Simulation Conference – WSC 2010*, pages 166–183.

Sulistio, A., Cibej, U., Venugopal, S., Robic, B., and Buyya, R. (2008). A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice and Experience*, 20(13):1591–1609.

Taylor, I. J., Deelman, E., Gannon, D. B., and Shields, M. (2006). *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Yu, J. and Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3):44–49.

Yu, Z. and Shi, W. (2007). An Adaptive Rescheduling Strategy for Grid Workflow Applications. In *IEEE International Parallel and Distributed Processing Symposium, 2007 – IPDPS 2007*, pages 1–8.

Anexo C | Una solución SOA para ejecutar *workflows* científicos en entornos Grid heterogéneos

En este Anexo se muestra el artículo titulado "Una solución SOA para ejecutar workflows científicos en entornos Grid heterogéneos" el cual ha sido aceptado para su publicación en las VIII Jornadas de Ciencia e Ingeniería de Servicios (JCIS 2012) que se celebrará en Almería (España), del 17 al 19 de Septiembre de 2012.

El artículo se presenta manteniendo el formato con el que aparecerá en el congreso mencionado.

Una solución SOA para ejecutar *workflows* científicos en entornos Grid heterogéneos

Sergio Hernández, Javier Fabra, Pedro Álvarez, and Joaquín Ezpeleta

Instituto de Investigación en Ingeniería de Aragón (I3A)
Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España
{shernandez,jfabra,alvaper,ezpeleta}@unizar.es

Abstract. La posibilidad de ejecutar un mismo *workflow* científico en distintos entornos Grid heterogéneos es todavía a día de hoy un reto abierto. Aunque la orientación a servicios logró allanar el camino, las propuestas de solución existentes aún requieren un papel activo por parte de los programadores de los *workflows*. En este trabajo se pretende dar un paso más allá, liberando al programador de esta responsabilidad. Concretamente, se propone un servicio de computación que permite programar *workflows* independientemente del entorno de ejecución y a diferentes niveles de abstracción. El servicio integra diversas infraestructuras Grid heterogéneas, sacando el máximo provecho de las mismas mediante una estrategia de *meta-scheduling* basada en simulación. Como caso de uso, el *workflow* de análisis *Inspiral* es ejecutado sobre dos Grids mejorando el rendimiento del *workflow*.

Keywords: *Workflows* científicos, orientación a servicios, *Grid*, integración de sistemas heterogéneos

1 Introducción

El creciente interés de la comunidad científica por automatizar de manera sistemática la ejecución de sus experimentos ha supuesto el impulso definitivo de los *workflows* científicos. Este tipo de *workflow* presenta unas características muy particulares que condicionan su ejecución: están compuestos por actividades complejas desde el punto de vista de los recursos computacionales necesarios para su ejecución, gestionan grandes volúmenes de datos como entrada/salida de las tareas ejecutadas, y necesitan gestionar adecuadamente la disponibilidad de recursos *hardware* y *software* heterogéneos. Por otro lado, el paradigma de computación Grid [1] propone el desarrollo de infraestructuras de computación formadas por recursos heterogéneos y geográficamente distribuidos. La capacidad computacional y las comunicaciones en red de este tipo de infraestructura han promovido su explotación como entornos para la ejecución de *workflows* científicos.

La programación de *workflows* científicos ejecutables en este tipo de infraestructuras Grid ha experimentado una fuerte evolución. Ésta está

condicionada por el nivel de abstracción ofrecido por los *middlewares* construidos para gestionar este tipo de infraestructuras, como gLite [2] o Condor [3]. Inicialmente, cada *middleware* concreto especificaba su propio lenguaje de programación de bajo nivel. Este enfoque tiene dos problemas. Primero, el programador debe conocer en detalle las características intrínsecas del *middleware* (configuración *hardware* y *software*, mecanismos de interacción, recursos disponibles, etc.), en vez de preocuparse únicamente por los aspectos funcionales del *workflow* a programar. Segundo, los *workflows* resultantes están altamente acoplados al *middleware* utilizado y, en muchos casos, a la infraestructura de computación concreta sobre la que se van a ejecutar.

En respuesta a estos dos problemas emerge la orientación a servicios desde dos direcciones diferentes [4]. Por un lado, nacen los denominados sistemas de gestión de *workflows* como herramientas de ayuda al programador (Taverna [5] o Kepler [6], entre otros). Las tareas de un *workflow* pueden ahora ser programadas como invocaciones a servicios, abstrayendo los detalles de bajo nivel relacionados con su futura ejecución. Por otro lado, los *middlewares* Grid ofrecen su funcionalidad a través de servicios, bajo las premisas y recomendaciones del estándar OGSA [7]. El uso combinado de sistemas de gestión y *middlewares* orientados a servicios fue clave para liberar al programador de los detalles de bajo nivel, resolviendo el primero de los problemas mencionados. Sin embargo, no se logró desacoplar completamente los sistemas de gestión de *workflows* de los *middlewares*. Este hecho dificulta que un *workflow* programado con una herramienta concreta sea portable y ejecutable en *middlewares* heterogéneos.

Los portales surgen como una alternativa de solución. Heredan las virtudes de programar *workflows* con una orientación a servicios y añaden la posibilidad de usar distintas infraestructuras de computación a través de una interfaz única. Algunos ejemplos de portales son P-GRADE [8] o HPC-Europa [9]. Aunque constituyen un avance en el problema de ejecutar un mismo *workflow* en distintas infraestructuras, presentan un problema de flexibilidad relativo a su proceso de *scheduling*: determinar en qué recursos computacionales se ejecuta cada tarea concreta. En estos portales el *scheduling* es estático y guiado por el usuario. Esto implica que el usuario es el encargado de seleccionar la infraestructura de ejecución al comienzo del *workflow*, sin que esta información pueda ser modificada durante su ejecución. Esta limitación provoca que no se logre un buen aprovechamiento de los recursos disponibles, ya que el proceso de *scheduling* es muy complejo, y el usuario no suele disponer de información que le ayude en la toma de la decisión. Como consecuencia, las tareas que forman los *workflows* se ven sometidas a elevados tiempos de espera, con la consiguiente reducción del rendimiento obtenido.

En este trabajo pretendemos avanzar un paso más allá. El objetivo es lograr que los *workflows* puedan ser ejecutados en diferentes *middlewares* e infraestructuras de computación, de forma que el usuario o programador no deba preocuparse en ningún momento de esta heterogeneidad. Para conseguirlo se define un servicio de computación que permite la ejecución de *workflows* científicos y tareas computacionalmente muy costosas cumpliendo las

características anteriores. El servicio encapsula e integra dinámicamente distintas infraestructuras de computación heterogéneas. Además, es capaz de determinar cuál es la infraestructura disponible más adecuada para la ejecución de cada tarea en base a un mecanismo de *meta-scheduling* basado en simulación, liberando de esta responsabilidad al programador. Finalmente, la interfaz del servicio facilita su uso conjunto con los sistemas de gestión de *workflows* científicos existentes, lo que facilita su utilización por diferentes usuarios acostumbrados a diversas herramientas y lenguajes de modelado.

El resto de este artículo se organiza como sigue. En la Sección 2, se realiza una descripción del servicio indicando su interfaz y cómo utilizar el mismo. En la Sección 3, se muestra la arquitectura interna del servicio y se explican las componentes fundamentales del mismo. La Sección 4 muestra el proceso de *meta-scheduling* utilizado para seleccionar la infraestructura de ejecución en la que ejecutar las tareas enviadas al servicio. La aplicación del servicio a un caso real se detalla en la Sección 5, utilizando como caso de estudio el *workflow* de análisis LIGO *Inspiral*. Finalmente, en la Sección 6, se presentan las conclusiones.

2 Descripción del servicio de computación

El servicio de computación ofrecido tiene como objetivo permitir la ejecución de *workflows* científicos y tareas computacionalmente muy costosas en entornos de computación Grid. La Figura 1 refleja las diferentes posibilidades existentes para utilizar el servicio.

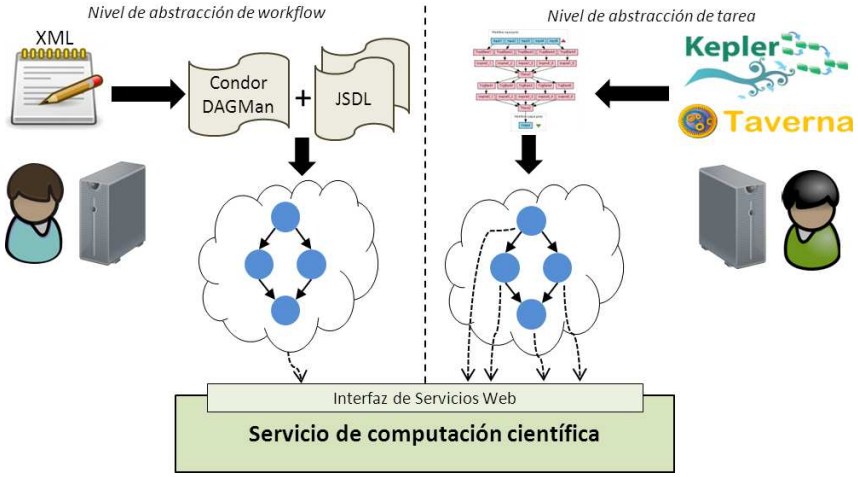

Fig. 1. Interacción de los programadores de *workflows* con el servicio de computación ofrecido a diferentes niveles de abstracción.

El servicio permite trabajar a dos niveles de abstracción: nivel de *workflow* y nivel de tarea. El nivel de abstracción de *workflow* permite solicitar al servicio la ejecución completa de un *workflow*, de forma que el servicio se encarga de la gestión de todo el ciclo de vida del mismo, liberando al usuario de esta labor. Esta alternativa corresponde a la parte izquierda de la Figura 1. Por su parte, trabajar a nivel de abstracción de tarea, permite al programador utilizar sistemas de gestión que controlen el ciclo de vida del *workflow* y solicitar la ejecución de las tareas que componen el mismo bajo demanda. Esta alternativa, reflejada en la parte derecha de la Figura 1, proporciona más flexibilidad al programador y permite integrar en el *workflow* tareas locales y servicios externos.

En caso de que el programador desee trabajar a nivel de *workflow*, tan sólo es necesaria la utilización de un editor de texto para describir las tareas y las relaciones existentes entre las mismas. Para describir las tareas se utiliza el estándar *Job Submission Description Language* (JSDL), mientras que para describir las relaciones existentes entre las mismas se usa el lenguaje de CondorDAGMan.

JSDL [10] es un lenguaje estándar propuesto por el *Open Grid Forum*[1] para la descripción textual de tareas, utilizando una sintaxis XML. El mismo ha sido ampliamente utilizado en entornos de computación Grid. La descripción de los trabajos incluye la aplicación a ejecutar, los argumentos pasados a la aplicación, referencias a los datos de entrada y salida involucrados (representados por las URIs correspondientes) y una descripción de los recursos necesarios para ejecutar la tarea (Sistema Operativo, arquitectura de la máquina, número de CPUs, memoria necesaria, ancho de banda de la red, etc.). Nótese que la descripción de los recursos necesarios no implica que el usuario tenga que especificar los recursos concretos de ejecución, si no que permite indicar las características que deben cumplir dichos recursos para poder ejecutar las tareas.

Por su parte, CondorDAGMan [3] es un sistema de gestión de *workflows* que permite especificar de forma textual la relación existente entre las tareas de un *workflow*. Mediante este lenguaje, se pueden especificar de una forma sencilla las relaciones existentes entre las tareas.

Si por el contrario, el usuario trabaja a nivel de tarea, el servicio puede utilizarse de forma conjunta con un sistema de gestión de *workflows*. La utilización de este tipo de sistemas está muy extendida, en parte debido a que hay sistemas de gestión orientados a una determinada comunidad científica, de forma que presentan facilidades a la hora de construir *workflows* de ese tipo (es el caso de Taverna que está enfocado en bioinformática). En esta alternativa, el programador especifica las relaciones existentes entre las tareas del *workflow* utilizando el lenguaje proporcionado por el sistema de gestión y envía las tareas al servicio individualmente, siendo el sistema de gestión el encargado de controlar el flujo de ejecución del *workflow*.

En cuanto a la funcionalidad ofrecida, el servicio permite la ejecución de *workflows* programados de forma completamente independiente del entorno de ejecución. Asimismo, los *workflows* pueden programarse a diferentes niveles

[1] http://www.ogf.org/

de abstracción proporcionando una gran flexibilidad al programador. Esta flexibilidad es necesaria por el elevado número de sistemas de gestión utilizados actualmente y la gran diversidad de usuarios existentes, los cuales no sólo están acostumbrados a diferentes herramientas, sino también a diferentes lenguajes y modelos de interacción. Para permitir diferentes tipos de interacción, el servicio incorpora internamente varios componentes de gestión que permiten controlar y gestionar adecuadamente el ciclo de las tareas a ejecutar independientemente del nivel de abstracción utilizado por el usuario (véase la Sección 3). Por otro lado, la necesidad de integrar y utilizar conjuntamente diversas infraestructuras de computación heterogéneas y componentes que permitan utilizar las mismas de forma adecuada y obtener el mayor rendimiento posible (véase la Sección 4).

Para soportar los diferentes tipos de utilización ofrecidos, se ofrecen mecanismos de interacción tanto síncronos (bloqueantes) como asíncronos (no bloqueantes). Asimismo, se incluyen operaciones que permiten la monitorización de los *workflows* y la obtención de los resultados junto con el log de ejecución de los mismos. En el caso de utilizar un modelo de comunicación síncrono, las operaciones ofrecidas por el servicio son las siguientes:

- *execWorkflowS*: Ejecuta de forma completa un *workflow*. Recibe como parámetros la descripción de las tareas y las dependencias existentes entre las mismas y devuelve el log de la ejecución de las tareas y referencias a los resultados.
- *execTaskS*: Ejecuta una tarea. Recibe como parámetro la descripción de una tarea y devuelve el log de ejecución de la tarea y referencias a los resultados.

En caso de utilizar un modelo de comunicación asíncrono, las operaciones ofrecidas por el servicio son las siguientes:

- *execWorkflowA*: Ejecuta de forma completa un *workflow*. Recibe como parámetros la descripción de las tareas y las dependencias existentes entre las mismas y devuelve como resultado un identificador del *workflow*.
- *execTaskA*: Ejecuta una tarea. Recibe como parámetro la descripción de una tarea y devuelve como resultado un identificador de la tarea.
- *getStatus*: Obtiene el estado de una tarea o *workflow*. Recibe como parámetro el identificador de una tarea o *workflow* y devuelve como resultado el estado de la tarea o *workflow* correspondiente.
- *getResult*: Obtiene el resultado de una tarea o *workflow*. Recibe como parámetro el identificador de una tarea o *workflow* y devuelve como resultado su log de ejecución y referencias a los resultados.

Adicionalmente, las operaciones asíncronas permiten indicar una dirección de correo electrónico para avisar al usuario de que la ejecución ha finalizado y facilitar el seguimiento de su estado y la recogida de los resultados.

Para la implementación del servicio se ha utilizado tecnología de Servicios Web por ser estándar en la construcción de *middlewares* Grid [7], ser sencilla de utilizar y presentar la suficiente flexibilidad para dar soporte a los diferentes tipos de interacción propuestos. Concretamente, se han desarrollado interfaces SOAP, junto con WSDL (para la descripción de las operaciones), y REST.

3 Arquitectura interna del servicio

En esta sección, se presenta la arquitectura interna del servicio mostrando sus principales componentes. La Figura 2 muestra dicha arquitectura. En la parte superior, se refleja la interacción con el programador del *workflow*, la cual se detalló en la sección anterior; en el centro de la figura se muestra la arquitectura interna del servicio; y en la parte inferior se indican las diferentes infraestructuras de computación integradas en el servicio junto con el *middleware* encargado de su gestión. A continuación, analizaremos en profundidad la arquitectura interna del servicio y describiremos las infraestructuras de computación utilizadas.

Internamente, el servicio también utiliza una arquitectura SOA, siguiendo un modelo ESB, que se traduce en un diseño flexible en el cual las diferentes componentes se encuentran desacopladas y pueden ser sustituidas, adaptadas o modificadas de forma dinámica y transparente para el usuario. Concretamente, el servicio está formado por un *broker* de recursos y un conjunto de componentes de gestión. El *broker* constituye el núcleo del servicio, encargándose de gestionar la interacción con el exterior, conocer el estado de las infraestructuras y permitir la comunicación entre los diferentes componentes del sistema. Por su parte, las componentes de gestión ofrecen diferentes funcionalidades encaminadas a la gestión del ciclo de vida de los *workflows* y a la mejora del mismo mediante, por ejemplo, la utilización de un *meta-scheduler* que permite decidir cuál es la mejor infraestructura para ejecutar una tarea.

A su vez, el *broker*, está formado por un repositorio de mensajes y una infraestructura de mediadores. La comunicación entre los diferentes componentes del servicio se realiza a través de mensajes que contienen información de diversa naturaleza. Por su parte, los mediadores encapsulan la heterogeneidad de un *middleware* determinado, teniendo completo conocimiento de sus capacidades y características. Este diseño elimina la necesidad de que el *broker* tenga que estar muy acoplado con la tecnología concreta de la infraestructura Grid, permitiendo incorporar diferentes infraestructuras heterogéneas de forma sencilla y transparente para el programador de *workflows*. Asimismo, permite la integración y sustitución dinámica de diferentes componentes de gestión.

En cuanta a la estructura interna del *broker*, por una parte, la implementación del repositorio de mensajes se ha inspirado en el modelo de coordinación Linda [11]. Los mensajes se codifican como tuplas y son almacenados en un espacio de tuplas. La interfaz del repositorio proporciona una serie de operaciones para acceder a las tuplas almacenadas de acuerdo a la semántica de Linda. Por otra parte, la infraestructura de mediadores permite lograr la integración de diferentes entornos de computación heterogéneos. En general, un mediador es una entidad que se comunica con el repositorio de tuplas, empareja y recupera las tuplas destinadas al mismo, de acuerdo a etiquetas identificativas presentes en las tuplas (tupla con tarea a ejecutar, tupla de fallo, etc.) y las procesa. En nuestro enfoque, un mediador representa un *middleware* capaz de gestionar una infraestructura de computación. Internamente, el mediador es responsable de: i) tener información completa de la infraestructura Grid que encapsula; ii) simular la ejecución de tareas en la misma; iii) interaccionar con el repositorio de tuplas para obtener

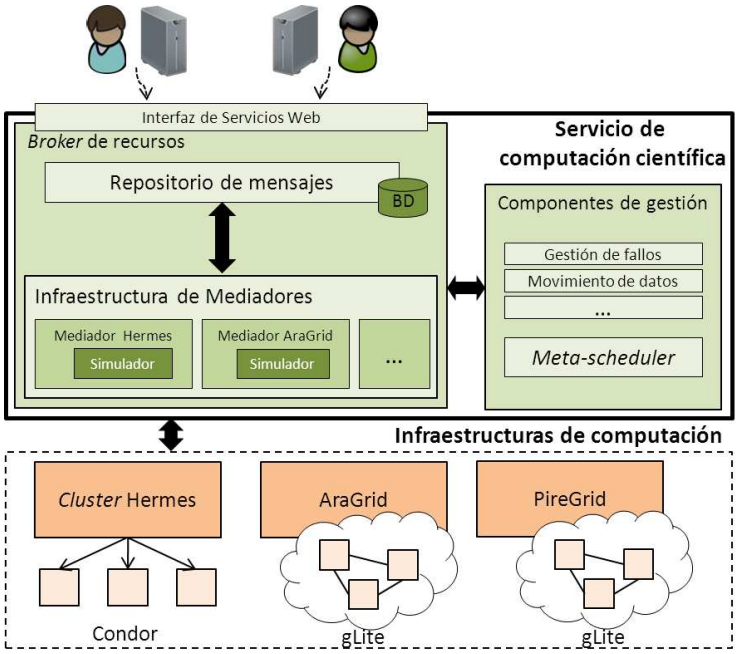

Fig. 2. Arquitectura del servicio propuesto e interacción del mismo con el exterior.

tareas a ejecutar o simular; iv) enviar tareas al *middleware* para su ejecución y controlar la transferencia de los datos de entrada y de salida; y v) insertar tuplas en el repositorio de mensajes con el resultado de la ejecución o simulación de las mismas para que esta información sea tratada por la componente adecuada. Se ha implementado un mediador para cada uno de los *middlewares* (Condor y gLite) utilizados por las infraestructuras de computación disponibles. Además, se ha incluido un simulador dentro de los mediadores para permitir realizar simulaciones que ayuden a decidir la infraestructura de ejecución más adecuada.

En cuanto a las componentes de gestión, éstas ofrecen diferentes funcionalidades encaminadas a gestionar el ciclo de vida de los *workflows* ejecutados. Se han desarrollado: una componente de gestión de fallos, una componente de movimiento de datos y una componente de *meta-scheduling* avanzado. El procedimiento de integración de estas componentes es similar al utilizado en los mediadores. Cada componente de gestión interacciona con el repositorio de mensajes para retirar mensajes con la etiqueta asociada a esa componente y procesarlos. Por lo tanto, la utilización de estas componentes puede ser debida a la necesidad de un procesado concreto (p. ej. textitmeta-scheduling) o como respuesta al resultado de otro componente (p. ej. gestión de fallos), permitiendo la composición dinámica de complejas cadenas de acción. Con la integración de estas componentes, se consigue gestionar de forma completa el ciclo de vida de un *worfklow* y se incluyen funcionalidades avanzadas como la utilización de un *meta-scheduler*

que permite obtener un mayor rendimiento y un mejor aprovechamiento de las infraestructuras disponibles. En la sección 4 se ofrecen más detalles sobre el *meta-scheduler*, mientras que en [12] se ofrecen más detalles sobre las componentes de movimiento de datos y gestión de fallos.

A modo de ejemplo, para que el lector comprenda la interacción existente entre las componentes del servicio, mostraremos el proceso seguido al utilizar la operación *execTaskA*, es decir, al ejecutar una tarea de forma asíncrona. En primer lugar, la descripción de la tarea se almacena en el repositorio de mensajes. A continuación, el *meta-scheduler* obtiene la tarea y solicita a los mediadores que simulen la ejecución de la misma. Con los resultados de las simulaciones, el *meta-scheduler* decide cuál es la infraestructura más adecuada para su ejecución. Dicha información es almacenada en el repositorio y recuperada por el mediador correspondiente. Antes de ejecutar la tarea, el mediador solicita el movimiento de los datos de entrada necesarios. Una vez transferidos, el mediador envía la tarea al Grid que representa para que se ejecute. Cuando la tarea finaliza o falla, el mediador solicita el movimiento de los datos de salida a su ubicación final, recupera el log de ejecución de la tarea e introduce dicha información en el repositorio de mensajes. Si la tarea ha finalizado correctamente, se envía un correo electrónico al usuario (sólo si se indicó al enviar la tarea) y la información queda almacenada en el repositorio de mensajes hasta que sea obtenida por el usuario a través de la operación *getResult*. Si por contra, la tarea ha fallado, la componente de gestión de fallos obtiene la causa del fallo y toma alguna decisión al respecto, por ejemplo, volver a ejecutar la tarea en otra infraestructura o notificar al usuario del error que se ha producido. En caso de que la tarea sea reejecutada, se repite el proceso, mientras que si se decide avisar al usuario del fallo, se actúa de la misma manera que en el caso de que la tarea finalice correctamente.

En lo que corresponde a las infraestructuras de computación, se han integrado: el *cluster* Hermes del Instituto de Investigación en Ingeniería de Aragón (I3A)[2], el cual es gestionado utilizando el *middleware* Condor; y dos Grids pertenecientes a la Iniciativa Grid Europea (EGI)[3]: AraGrid[4] y PireGrid[5], gestionados por el *middleware* gLite y administrados por el Instituto de Biocomputación y Física de Sistemas Complejos (BIFI)[6]. A pesar de su heterogeneidad, gracias a la infraestructura de mediadores y al diseño desacoplado de las componentes del sistema, su integración es sencilla y puede realizarse dinámicamente. Asimismo, la utilización de un *meta-scheduler* y otras componentes de gestión permite utilizar las infraestructuras de forma conjunta, siendo este proceso totalmente transparente para el usuario. Como resultado, se consigue dotar al servicio de una elevada potencia de cálculo y mejorar el rendimiento de los *workflows* ejecutados.

[2] http://i3a.unizar.es/
[3] http://www.egi.eu/
[4] http://www.araGrid.es/
[5] http://www.pireGrid.eu/
[6] http://bifi.es/es/

En resumen, la naturaleza abierta y flexible de la solución propuesta se basa en la utilización de un *broker* de recursos formado por un repositorio de mensajes basado en Linda y un conjunto de mediadores. El repositorio de mensajes facilita la integración de mediadores y componentes de gestión de forma dinámica, así como su sustitución, modificación y eliminación. La integración de un conjunto de mediadores encapsula la heterogeneidad de las diferentes infraestructuras de computación utilizadas, desacopla el *broker* de recursos de los detalles de los diferentes *middleware* de Grid existentes, abstrae al usuario de la complejidad de los mismos, permite la reutilización de mediadores en infraestructuras gestionadas por el mismo *middleware* y facilita la integración de nuevas infraestructuras. Finalmente, la integración de diferentes componentes de gestión permite mejorar la gestión del ciclo de vida de las tareas ejecutadas ofreciendo servicios de *meta-scheduling*, movimiento de datos y gestión de fallos.

4 *Meta-scheduling* basado en simulación

En esta sección se detalla el proceso utilizado por el servicio para seleccionar la infraestructura más adecuada para ejecutar las tareas enviadas. Debido a su importancia en el proceso, se prestará especial atención al componente de *meta-scheduling* y a los simuladores incluidos dentro de cada mediador Finalmente, se incidirá en la importancia del *workload* utilizado para realizar las simulaciones.

La introducción de una componente de *meta-scheduling* proporciona nuevas oportunidades respecto a estrategias de planificación básicas que obligan al usuario a indicar la plataforma de ejecución, seleccionan una infraestructura de forma aleatoria o elijen una infraestructura en base a criterios estáticos. Se han propuesto multitud de estrategias en la literatura [13]. En general, estas estrategias buscan optimizar algún tipo de función objetivo como, por ejemplo, el coste de utilización de los recursos o el tiempo de ejecución. En nuestro caso concreto, el algoritmo utilizado por la componente de *meta-scheduling* tiene como objetivo minimizar el tiempo de ejecución, utilizando para ello los resultados proporcionados por los simuladores. En cualquier caso, la discusión de la mejor estrategia de *meta-scheduling* posible queda fuera del alcance de este trabajo.

La Figura 3 muestra el proceso que supone la ejecución de las tareas de un *workflow* utilizando un *meta-scheduler*. Inicialmente, las tareas pendientes (abstractas), almacenadas en el repositorio de mensajes, son recuperadas por el *meta-scheduler*. A continuación, esta componente determina las infraestructuras capaces de ejecutar dichas tareas y solicita a los mediadores correspondientes que simulen su ejecución. Los mediadores realizan la simulación, utilizando *workloads* construidos dinámicamente, y devuelven el resultado al *meta-scheduler*. Cuando el *meta-scheduler* ha recibido el resultado de todas las simulaciones, elige la infraestructura más adecuada en base al algoritmo de optimización utilizado y almacena dicha información en la descripción de la tarea, la cual pasa a ser una tarea concreta. Finalmente, el *meta-scheduler* envía la tarea al repositorio de mensajes para que sea recuperada por el mediador correspondiente y ejecutada.

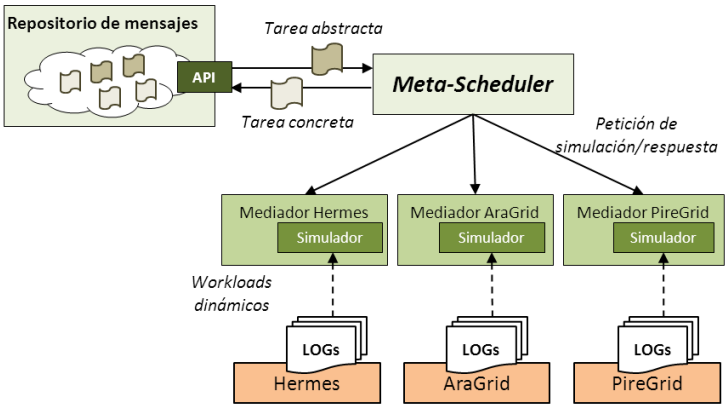

Fig. 3. Componente arquitectural para realizar *scheduling* basado en simulación.

Para soportar este proceso, los mediadores han sido extendidos mediante la integración de un simulador dentro del mismo. El simulador es capaz de: i) modelar el entorno de computación (recursos computacionales, memoria, ancho de banda de la red, usuario, política de *scheduling*, etc.); ii) seleccionar el *workload* más adecuado para representar el comportamiento real de cada entorno de ejecución y el estado actual de sus recursos (para crear estos *workloads* se han utilizado logs de ejecución reales); y, finalmente, iii) simular la ejecución de tareas midiendo parámetros como el tiempo de ejecución, el tiempo de transferencia de los datos, el tiempo de encolamiento, la memoria consumida, etc. La integración del simulador como componente interno de cada mediador permite que los mismos sean capaces de manejar diferentes situaciones y personalizar la simulación dependiendo del estado concreto de la infraestructura. En cualquier caso, el simulador es accedido a través de una API bien definida, de forma que añadir nuevos simuladores es sencillo y sólo implica modificar el modelo de la infraestructura y la política de *scheduling* utilizada por la misma .

Como base para los simuladores desarrollados se ha utilizado Alea [14]. Alea es un simulador basado en eventos y construido sobre GridSim [15]. Alea extiende GridSim proporcionando un *scheduler* centralizado, mejorando algunas funcionalidades y aumentando la escalabilidad y la velocidad de la simulación. Además, Alea proporciona un entorno de experimentación fácil de configurar y utilizar, el cual ayuda en la adaptación del simulador a las nuevas infraestructura añadidas al servicio. La implementación de Alea ha sido extendida para soportar tanto la política de scheduling basada en prioridades utilizada por *Condor* como la política jerárquica utilizada por *gLite*.

En lo que respecta a las simulaciones, un aspecto clave es la creación del *workload* que indica las tareas a simular. La importancia de utilizar un *workload* apropiado ha sido identificada en varios trabajos [16, 17]. El uso de un *workload* erróneo puede provocar que los resultados de la simulación no se ajusten al comportamiento real y, por tanto, lleven a tomar malas decisiones de *meta-*

scheduling. En nuestro caso, los *workloads* se crean dinámicamente mediante la agregación de las tareas que se quieren simular, las tareas que se están ejecutando actualmente en la infraestructura y una serie de tareas que representan la carga esperada de la infraestructura. Más detalles sobre la construcción de los *workloads* y los propios simuladores pueden encontrarse en [18].

En conclusión, la utilización de un *meta-scheduler* basado en simulación permite sacar partido de la integración de diversas infraestructuras heterogéneas, logrando una mejor utilización de los recursos disponibles, lo que se traduce en una disminución del tiempo de ejecución de los *worklows* ejecutados.

5 Caso de estudio: Inspiral

En esta sección desplegaremos y ejecutaremos el *workflow* científico de análisis *Inspiral* con la solución propuesta. El *workflow* de análisis Inspiral es un *workflow* científico que analiza e intenta detectar ondas gravitacionales producidas por varios eventos en el universo utilizando datos obtenidos de la coalescencia de sistemas binarios compactos como estrellas binarias de neutrones y agujeros negros [19]. La Figura 4 muestra la estructura del *workflow* (Figura 4-a) y su implementación en Taverna (Figura 4-b). Como puede observarse, la relación entre cada una de las tareas que forman una fase en el diseño de alto nivel y la implementación correspondiente del *workflow* en Taverna es inmediata, siendo muy sencilla la composición del experimento. Internamente, cada una de las cajas que representan las tareas en Taverna encapsula varias operaciones sencillas como son la generación de la descripción de las tareas y las invocaciones al servicio.

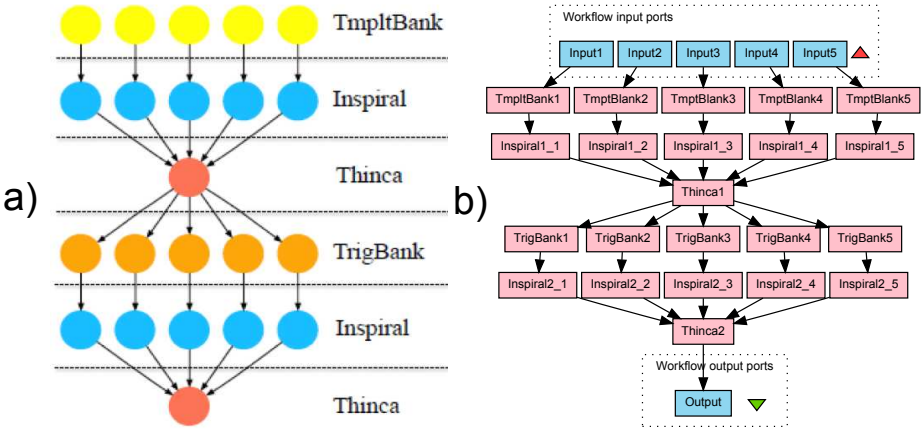

Fig. 4. Workflow científico de análisis LIGO Inspiral: a) Descripción de alto nivel, b) Implementación en Taverna.

El experimento consta de diferentes fases, cuya descripción detallada puede consultarse en [18, 19], que realizan el procesamiento de grandes conjuntos de mediciones generadas por un conjunto de sensores y detectores. Los trabajos *Inspiral* son los más complejos en términos computacionales y los que más recursos demandan.

El *workflow* Inspiral se puede ejecutar tanto en Hermes como en AraGrid. Sin embargo, ambas infraestructuras muestran una tendencia a tener diferentes niveles de carga a lo largo del día, lo que provoca que sea más viable enviar los trabajos a una infraestructura en ciertos momentos y viceversa. Los detalles de los diferentes niveles de carga pueden consultarse en [18]. Evidentemente, éste no es el único criterio a considerar ya que el rendimiento de una plataforma Grid depende de muchos factores y su análisis es complejo. La utilización de un simulador como herramienta de decisión permite lidiar con esta complejidad y mejorar el rendimiento obtenido en la ejecución del *workflow* como se muestra en la Figura 5. La figura muestra el tiempo de ejecución total de cada etapa para el *workflow Inspiral* ejecutado de forma completa en cada infraestructura (la barra izquierda corresponde a Hermes mientras que la barra derecha corresponde a AraGrid) y ejecutado utilizando el servicio y la política de *meta-scheduling* descrita anteriormente (barra central de la figura). Los resultados muestran que, para el caso del experimento Inspiral, la utilización del servicio permite obtener una mejora del 59% respecto a la ejecución del *workflow* en Hermes y un 111% respecto a la ejecución del mismo en AraGrid.

Respecto a la sobrecarga que introduce la simulación en términos de tiempo de ejecución, el proceso de simulación de Hermes es más complejo y tarda entre 3 y 4 minutos para una bolsa de 10000 tareas, mientras que para AraGrid lleva en torno a 1 minuto. Además, el tiempo de simulación es insignificante en comparación con el tiempo de ejecución de cada etapa y las transferencias de datos entre infraestructuras usan enlaces de alta velocidad lo que implica que el tiempo de ejecución disminuya a pesar de la sobrecarga introducida.

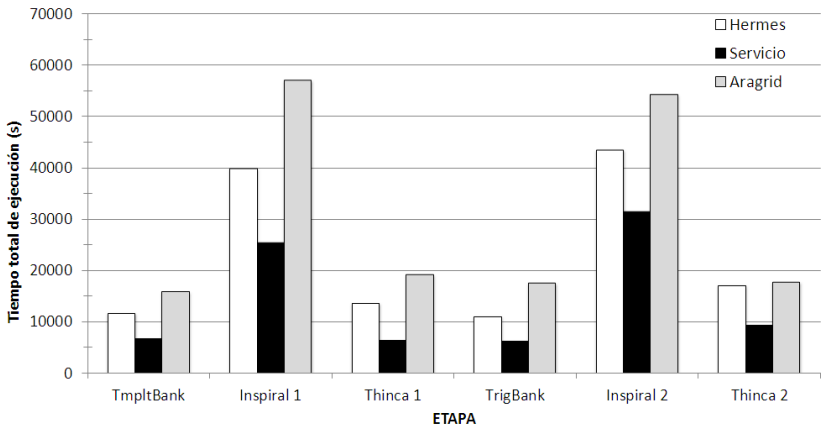

Fig. 5. Resultados experimentales para el *workflow* Inspiral.

6 Conclusiones

En este artículo se ha propuesto un servicio de computación para la ejecución de *workflows* científicos. El servicio soluciona varios de los problemas abiertos en el contexto de la ejecución de *workflows* científicos en infraestructuras Grid, en lo que atañe al modelado de los *workflows*y a la ejecución de los mismos.

En primer lugar, el servicio permite ejecutar *workflows* programados de manera independiente del entorno de ejecución. Esto permite liberar al programador de los detalles de bajo nivel referentes a la infraestructura y la interacción con el *middleware*, lo que facilita la creación del *workflow*. Asimismo, la posibilidad de utilizar el servicio junto con un sistema de gestión de *workflows* permite ejecutar *workflows* programados en diferentes lenguajes, facilitando el uso del servicio por usuarios con diferentes conocimientos y necesidades.

En segundo lugar, el diseño flexible y desacoplado propuesto permite integrar diversas infraestructuras de computación heterogéneas de forma dinámica y utilizar las mismas conjuntamente, dotando al servicio de una elevada potencia computacional. La inclusión de un *meta-scheduler* y técnicas de simulación permite decidir de forma dinámica la infraestructura más adecuada para ejecutar cada tarea, sacando el máximo partido posible a las infraestructuras disponibles y obteniendo una mejora en el rendimiento de los *workflows* ejecutados, como puede observarse en la aplicación del mismo al *workflow* de análisis *Inspiral*.

Actualmente, se está trabajando en la mejora de diferentes aspectos del servicio. Por un lado, se pretenden integrar más infraestructuras de computación, en particular se plantea la utilización de entornos de computación Cloud como Amazon EC2. Por otra parte, se pretende aumentar la precisión de las simulaciones, para poder tomar mejores decisiones y aumentar el rendimiento de los *workflows* ejecutados. En esa misma línea, se pretende estudiar el rendimiento que ofrecen diferentes algoritmos de *meta-scheduling*. Finalmente, se pretende aplicar el servicio a la resolución de problemas complejos desde un punto de vista computacional, como el análisis del comportamiento de *workflows* científicos anotados semánticamente.

Agradecimientos

Este trabajo se ha financiado parcialmente gracias al proyecto de investigación del Ministerio de Educación y Ciencia TIN2010-17095.

References

1. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2003)
2. Laure, E., Gr, C., Fisher, S., Frohner, A., Kunszt, P., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Byrom, R., Cornwall, L., Craig, M., Meglio, A.D., Djaoui, A., Giacomini, F., Hahkala, J., Hemmer, F., Hicks, S., Edlund, A., Maraschini, A., Middleton, R., Sgaravatto, M., Steenbakkers,

M., Walk, J., Wilson, A.: Programming the Grid with gLite. In: Computational Methods in Science and Technology. (2006)
3. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience: Research articles. Concurrency and Computation: Practice and Experience **17** (2005) 323–356
4. Foster, I.: Service-Oriented Science. Science **308** (2005) 814–817
5. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic acids research **34** (2006) W729–732
6. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience **18** (2006) 1039–1065
7. Gannon, D., Plale, B., Christie, M., Fang, L., Huang, Y., Jensen, S., Kandaswamy, G., Marru, S., Pallickara, S.L., Shirasuna, S., Simmhan, Y., Slominski, A., Sun, Y.: Service oriented architectures for science gateways on Grid systems. In: ICSOC. (2005) 21–32
8. Farkas, Z., Kacsuk, P.: P-GRADE Portal: A generic workflow system to support user communities. Future Generation Comp. Syst. **27** (2011) 454–465
9. Oleksiak, A., Tullo, A., Graham, P., Kuczynski, T., Nabrzyski, J., Szejnfeld, D., Sloan, T.: HPC-Europa: Towards uniform access to European HPC infrastructures. In: 6th IEEE/ACM International Conference on Grid Computing (GRID 2005), IEEE (2005) 308–311
10. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., Mcgough, S., Pulsipher, D., Savva, A.: Job Submission Description Language (JSDL) Specification, Version 1.0. Technical report, Global Grid Forum (2005)
11. Carriero, N., Gelernter, D.: Linda in context. Commun. ACM **32** (1989) 444–458
12. Fabra, J., Hernández, S., Álvarez, P., Ezpeleta, J.: A framework for the flexible deployment of scientific workflows in grid environments. In: To appear in The Third International Conference on Cloud Computing, GRIDs, and Virtualizations (CLOUD COMPUTING 2012). (2012)
13. Li, Y., Lan, Z.: A survey of load balancing in Grid computing. In: Proceedings of the First international conference on Computational and Information Science. CIS'04, Berlin, Heidelberg, Springer-Verlag (2004) 280–285
14. Dalibor Klusácek, H.R.: Alea 2 – job scheduling simulator. In: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), ICST (2010)
15. Sulistio, A., Cibej, U., Venugopal, S., Robic, B., Buyya, R.: A toolkit for modelling and simulating data grids: an extension to gridsim. Concurrency and Computation: Practice and Experience **20** (2008) 1591–1609
16. Feitelson, D.: Workload modeling for performance evaluation. In: Performance Evaluation of Complex Systems: Techniques and Tools. Springer, Berlin / Heidelberg (2002) 114–141
17. Li, H., Groep, D., Wolters, L.: Workload characteristics of a multi-cluster supercomputer, Springer Verlag (2004) 176–193
18. Hernández, S., Fabra, J., Álvarez, P., Ezpeleta, J.: A simulation-based scheduling strategy for scientific workflows. In: To appear in the Second International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2012). (2012)
19. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for e-Science: Scientific Workflows for Grids. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)