Resource provisioning for e-Science environments

Andrea Bosin
Dipartimento di Fisica, Università degli Studi di Cagliari, Italy
Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Cagliari, Italy

Pre-print submitted to International Journal of Grid and High Performance Computing (IGI Global)

ABSTRACT

Recent works have proposed a number of models and tools to address the growing needs and expectations in the field of e-Science. At the same time, the availability and models of use of networked computing resources needed by e-Science are rapidly changing and see the coexistence of many disparate paradigms: high performance computing, grid and recently cloud, which brings very promising expectations due to its high flexibility. In this paper we suggest a model to promote the convergence and the integration of different computing paradigms and infrastructures for the dynamic on-demand provisioning of the resources needed by e-Science environments, leveraging the Service-Oriented Architecture model. In addition, its design aims at endorsing a flexible, modular, workflow-based collaborative environment for e-Science. A working implementation used to validate the proposed approach is described together with some performance tests.

Keywords: e-Science; resource provisioning; Service Oriented Architectures; Workflows; BPEL; Web Services; HPC; Grid; Cloud.

INTRODUCTION

Recent works (Akram, 2006; Deelman, 2009; Elmroth, 2010; McPhillips, 2009) have proposed a number of models and tools to address the growing needs and expectations in the field of e-Science. In particular, Akram (2006) and Bosin (2011a) have shown the advantages and the feasibility, but also the problems, of modeling e-Science environments and infrastructures according to the Service-Oriented Architecture (SOA) and its enabling technologies such as Web Services (WS). Among the main advantages of such approach we find: interoperability, open standards, modularity, dynamic publish-find-bind, and programmatic access.

At the same time, the availability and models of use of networked computing resources needed by e-Science are rapidly changing and see the coexistence of many disparate paradigms, featuring their own characteristics, advantages and limitations. Among the main paradigms we find High Performance (HPC), Grid and Cloud Computing. In all cases, the objective is to best provide hardware and software resources to user applications with the help of schedulers, reservation systems, control interfaces, authentication mechanisms and so on. A detailed comparison of the characteristics of HPC, grid and cloud paradigms is presented by Mateescu (2011), which observes that none of these paradigms is the ultimate solution, and a convergence of HPC, grid and cloud resources should be pursued. Such a convergence must take into account

a number of differences between both resources and provisioning systems, which can be intrinsic but may also depend on the different user requirements:

- hardware capabilities,
- network connectivity,
- operating systems,
- middleware,
- software and libraries,
- application programming interface (API),
- authentication models, and
- usage models.

It is important to underline that such differences are essential in allowing the execution of applications with very different requirements, even if they may constitute an obstacle in the quest for the perfect computing paradigm. According to our experience, an important point in the usability of computing infrastructures available to e-Science is then to guarantee user access to the widest spectrum of resources in a technology agnostic way.

In this paper we suggest a model to promote the convergence and the integration of different computing paradigms and infrastructures for the dynamic on-demand provisioning of the resources needed by e-Science environments, leveraging the SOA model. At the same time, the design aims at endorsing a flexible, modular, workflow-based collaborative environment for e-Science. The latter sees the integration and inter-operation of a number of software components, such as:

- workflows, to define and coordinate complex scientific application or experiments;
- service interfaces, to expose the business logic of scientific applications; and
- components, to implement business rules and perform business tasks related to a specific scientific domain.

At the implementation level, the choice of SOA as the enabling technology for a common integration and inter-operation framework sounds realistic due to

- availability of SOA standards for workflow systems;
- availability of web service libraries to build new applications and to wrap existing ones;
- existence of SOA standards covering areas like data access, security, reliability, etc.; and
- access to a number of computing infrastructures (e.g. grid) is, at least partially, SOA-aware.

Our model is not meant to replace existing HPC, grid and cloud paradigms, rather it is an attempt aimed at complementing, integrating and building on them by playing the role of a dynamic resource aggregator exposing a technology agnostic abstraction layer.

Our proposal borrows many SOA concepts and standards from the business domain, including the adoption of the Business Process Execution Language (BPEL) for workflow design and execution. A motivation of our approach is almost evident: the SOA paradigm, and in particular web services and BPEL, are based on widely accepted standards and supported by many software tools, both open source and commercial. In addition, an important feature of BPEL, essential in highly dynamic environments, is the capability of setting at run-time the network endpoint of the services to be invoked. This allows BPEL to work with services which may be brought into existence even after the workflow has started (e.g. by the workflow itself) and disposed before the workflow has ended (e.g. again by the workflow). However, SOA and WS do not preclude the use of other technologies and tools: the point is not whether they can be

adopted by e-Science environments, but if this can be done in such a way to allow the convergence and integration of a potentially enormous number of distributed resources, belonging to different organizations.

Our claim is not that SOA is the best or the only way to contribute to the development of e-Science collaborative environments, rather we wish to explore and suggest a new direction, moving from specialized systems towards open-standards service-oriented systems.

In addition we describe a working proof-of-concept implementation whose main objectives are to:

- validate the effectiveness of the model;
- assess its limitations;
- gain a better understanding of practical details and problems; and
- execute some simple performance measurements.

In the next section we review some related work, then provide a simple classification of resources, and present a summary of the existing resource provisioning systems, followed by the description a dynamic on-demand model promoting the convergence and the integration of such systems. Then, we give a description of a working implementation of the model, some details about authentication and security, and discuss some performance results, problems and possible solutions. Finally, we draw some conclusions.

RELATED WORK

The approach presented in this work has been previously pursued by Bosin (2011a) and Bosin (2011b), with particular emphasis on SOA and BPEL adequacy for e-Science environments, and less attention to the general problem of resource provisioning.

Mateescu (2011) presents a hybrid computing model which is aimed at executing scientific applications in such a way to satisfy the given timing requirements. Applications are represented by workflows, i.e. a set of jobs with precedence constraints. The basic building block is the Elastic Cluster, characterized by (1) dynamic infrastructure management services; (2) cluster-level services such as workload management; and (3) intelligent modules that bridge the gap between cluster-level services and dynamic infrastructure management services. An infrastructure for the management and execution of workflows across multiple resources is then built by using multiple Elastic Clusters coordinated by a workflow management system.

Dörnemann (2009) explores on-demand provisioning of cloud resources directly at the workflow level, using BPEL as the workflow language. When, during workflow enactment, a service is invoked, the request is routed to the service instance running on the best-matching host (e.g. lowest load); if no best-matching host is available, a new VM is provisioned from a cloud to run a new instance of the required service.

The idea of virtual clusters on a physical grid is explored by Murphy (2010), where Virtual Organization Clusters provide customized, homogeneous execution environments on a per-Virtual Organization basis. The author describes a clustering overlay for individual grid resources, which permits Virtual Organizations of federated grid users to create custom computational clouds with private scheduling and resource control policies.

The feasibility of using one or more cloud providers for deploying a grid infrastructure or parts of it has been studied by Vàzquez (2011); such an approach permits the elastic growth of the given grid infrastructure in such a way to satisfy peak demands or other requirements.

The lack of a standard architecture for resource provisioning is discussed in (Mietzner, 2008), where the authors present a set of services for resource procurement and BPEL-based workflows that make use of these services.

McPhillips (2009) identifies desiderata for scientific workflow systems - namely clarity, predictability, reportability and reusability. Moreover, ease of composition and editing, the ability to automatically log and record workflow enactments and the flexibility to incorporate new tools are all important features (Fox, 2006). The interoperability aspects of scientific workflow systems are addressed by Elmroth (2010), which investigates the differences in execution environments for local workflows and those executing on remote grid resources. A complete overview of features and capabilities of scientific workflow systems is presented in (Deelman, 2009).

Software applications have been built to address a wide spectrum of scientific workflows, ranging from basic tools that are designed to handle "desktop" tasks such as simple data analysis and visualization to complex workflow systems that are designed to run large-scale e-Science applications on remote grid resources. These systems need to support multiple concurrent users, deal with security requirements, and run workflows that may require the use of a sophisticated layer of services (Fox, 2006).

There is a number of widely recognized grid workflow projects like Triana (Taylor, 2005; Churches, 2006), Kepler (Pennington, 2007), Pegasus (Deelman, 2005), and ASKALON (Fahringer, 2007). Many of these began their life in the "desktop" workflow space, and have evolved over time to address the large-scale e-Science applications. Specifically designed for the life sciences, Taverna (Oinn 2004, 2007) was the first system to recognize the importance of data provenance and semantic grid issues.

While developed for the business domain, technologies like BPEL are recognized suitable to address the requirements of e-Science applications (Akram, 2006), supporting the composition of large computational and data analysis tasks that must execute on supercomputing resources. BPEL is recognized by Deelman (2009) as the de facto standard for web-service-based workflows. An architecture for the dynamic scheduling of workflow service calls is presented in (Dörnemann, 2009), where control mechanisms of BPEL are combined with an adaptive runtime environment that integrates dedicated resources and on-demand resources provided by infrastructures like Amazon Elastic Compute Cloud. Ma (2008) presents the design and implementation of a workflow management system based on BPEL in a grid environment. Based on BPEL, QoWL (Brandic, 2006) and GPEL (Slominski, 2007) are significant examples of workflow systems designed for dynamic, adaptive large-scale e-Science applications.

The use of BPEL for grid service orchestration is proposed as foundation in (Leymann, 2006) since it fulfils many requirements of the WSRF standard. The appropriateness of BPEL is also examined and confirmed in (Chao, 2004; Dörnemann, 2007; Emmerich, 2005). These works mainly focus on scientific workflows and rely on extending or adapting BPEL, thus creating dialects.

RESOURCES

Personal resources, such as desktops or laptops, provide users with the maximum ease-of-use in terms of flexibility and customization, often at the expense of limiting the interoperability with other resources. A personal resource usually is the main door for accessing the network, and it is both the starting and end point for user interaction with e-Science environments (e.g. start an experiment, monitor its execution, collect results, write reports). *Specialized resources* are

“unique” nodes on the network where a very specific activity can take place (e.g. data acquisition from an attached physical device), but sharing such resources on the network may not be straightforward, since in many cases they expose custom APIs.

High Performance Computing (HPC) resources are tightly-coupled (e.g. by a high performance communication device such as an Infiniband switch) sets of computing equipment with (usually) a high degree of hardware and software homogeneity. HPC resources are in most cases managed by legacy schedulers which represent a big issue for interoperability.

Grid resources are loosely-coupled sets of computing equipment with (usually) a local (e.g. per site) high degree of hardware and software homogeneity. Physical separation is not an issue and resources are managed through the abstraction of Virtual Organizations (VO) with the possibility of federating many different VO as in project Enabling Grids for E-sciencE (EGEE, 2010). The Open Grid Services Architecture (Foster, 2002), or OGSA, is a framework for developing grid management services according to the Service-Oriented Architecture, thus enabling ease of integration and interoperability between sites, even if grids have limited interoperability between different Grid software stacks (Mateescu, 2011).

One level of abstraction higher, *cloud resources* are based on loosely-coupled sets of computing equipment with no need or guarantee of hardware and software homogeneity, and in many cases distributed at different physical locations. In this paradigm, the physical machine (PM) hardware and software is almost completely hidden, and a virtual machine (VM) abstraction is exposed to the user. Most cloud management systems are designed according to SOA concepts and expose their functionality through SOAP-based or RESTful interfaces thus improving ease of access, integration and interoperability. One interesting characteristic of clouds is the ability to provide resources with certain characteristics, i.e. specific software libraries or configurations, dynamically, on-demand and on a per-user basis (Vàzquez, 2011).

Even if many computing paradigms show some kind of SOA-awareness, heterogeneous resource provisioning still remains an open problem because of the many integration issues between paradigms. A brave user wishing to run a distributed application (eventually workflow-based) using a mix of heterogeneously managed resources may encounter a number of interoperability issues due to differences in:

- usage model,
- life-cycle management,
- authentication,
- API,
- adopted standards (or no standard at all),
- services (e.g. storage),
- network segmentation or isolation between different resources,
- monitoring, and
- workflow languages and engines.

A solution suitable to a wide community of users wishing to exploit at best the resources they have access to (e.g. those granted to them at no charge) is to design an abstraction layer between applications and resources, in such a way that resources can be requested, provisioned, used, monitored, and disposed without worrying about the underlying technological infrastructure. Such an approach will be explored later, suggesting a possible way to manage resources from multiple providers as a cohesive aggregate. Promoting the integration, cooperation and inter-operation of heterogeneous resources has the advantage of allowing users to exploit the best of each paradigm. To give some examples, grid infrastructures may offer:

- large sets of computing resources,
- resource scheduling/reservation,
- advanced storage services,
- specialized workflow systems,
- advanced monitoring services, and
- homogeneous operating system and software environments;

while HPC infrastructures typically provide:

- high performance hardware enabling strongly-coupled parallelism, and
- resource scheduling/reservation,
- homogeneous hardware and software environments;

and cloud infrastructures present:

- extremely flexible/customizable operating system and software environments,
- API based on open standards (e.g. SOAP, REST),
- on-demand provisioning, and
- increasing number of resources offered by commercial providers.

The availability of cloud resources, capable of instantiating standard or custom VM on a per-user basis, gives a number of value-added enhancements:

- allow the execution of platform-dependent applications which may be bound to specific operating system flavors or libraries not generally available;
- permit dynamic provisioning of different workflow engines; and
- give the additional flexibility to run web service applications and enact workflows where and when the user wishes (e.g. users may well decide to take advantage of pay-per-use commercial providers such as Amazon or RackSpace).

In a scenario based on "smart" resource aggregation, new applications may enjoy many of the above benefits; it is possible to imagine a distributed application orchestrated by a workflow where (1) a VM on a cloud runs the workflow engine, (2) the bulk computation is performed in parallel on a HPC infrastructure, (3) data post-processing is executed on a grid (or cloud), (4) results are viewed on a personal resource and, optionally, (5) steps 2-4 are part of an iterative computation.

At a first sight, there may be no benefits for existing applications, unless they are modified, but cloud flexibility may nevertheless help. Indeed, a user application developed for a grid or other resource may become unusable if (1) the user has no (or no longer) access to the grid, or (2) an update in grid operating system/middleware requires modifications to the application. A possible solution is then to deploy the desired flavor of grid on top of a cloud, as explored in (Vàzquez, 2011), and run the unmodified application.

PROVISIONING SYSTEMS

HPC and grid resources are usually controlled by *Workload and Resource Management Systems (WRMS)* which support

- resource management;
- job management; and
- job scheduling.

The term job refers to the set of instructions needed to run an application or part of it on a batch system, written according to some Job Description Language (JDL). A WRMS can be built on top of another WRMS, as it is usually the case with grid middleware which relies upon an underlying batch system referred to as a Local Resource Management System (LRMS).

Users are granted access to a WRMS in a number of ways:

- interactively from a front-end machine using a command line interface (CLI);
- programmatically by means of specialized API; and
- in some cases, programmatically by means of technology agnostic API, e.g. Open Grid Forum Distributed Resource Management Application API (DRMAA) (Tröger, 2011).

Authentication mechanisms may range from simple credentials such as user/password to sophisticated X.509 certificates based on a public key infrastructure, both for users and software services. Job submission to a grid WRMS is shown in Fig. 1 for gLite CREAM (Aiftimiei, 2010; Laure, 2006).

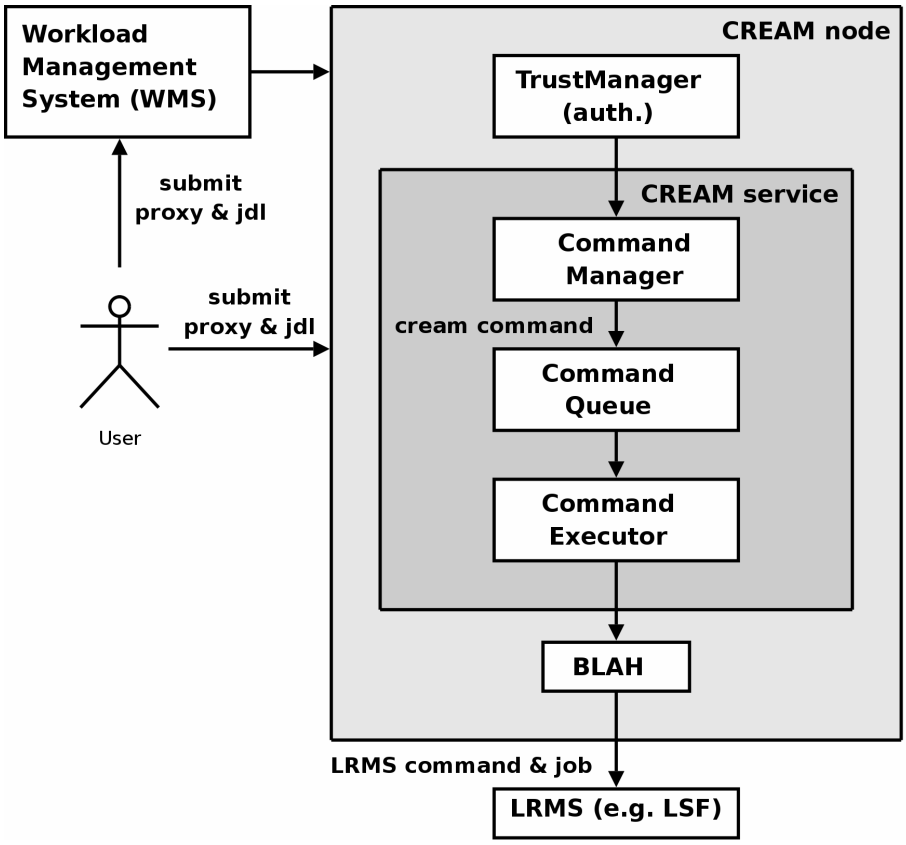

Figure 1. Job submission with gLite CREAM (simplified).

Job execution with CREAM can be summarized as follows: (1) A grid user, using his/her X.509 credentials, obtains an authentication proxy from the VO membership service (VOMS), prepares the job and submits both the JDL file and proxy to CREAM service; (2) user authentication and authorization (by Local Centre Authorization Service, LCAS) is performed; (3) user request is mapped to a cream command and queued; (4) the cream command is processed and mapped to a specific LRMS request and grid user is mapped to a LRMS user (by Local Credential Mapping Service, LCMAPS); and (5) the request is submitted to the LRMS.

CREAM is currently released only for Red Hat Enterprise Linux 5.x and derived distributions like CentOS 5.x and Scientific Linux 5.x, and supports the following LRMS: Platform Load

Sharing Facility (LSF, 2005), Oracle Grid Engine (SGE, 2009) and Adaptive Computing Torque/Maui (TORQUE, 2012).

Cloud resources are typically supervised by *Dynamic Infrastructure Management Systems (DIMS)*, which offer two kinds of functionality: (1) physical resource management; and (2) service management, where provisioned services are implemented using virtual resources. We are mainly interested in the *Infrastructure as a Service (IaaS)* model, which offers the maximum flexibility by providing virtual machines and the management interface, as well as storage. User access is performed:

- interactively from a client application;
- through a browser by means of suitable plug-ins;
- programmatically by means of specialized API; and
- programmatically by means of technology agnostic API such as Amazon Elastic Compute Cloud (EC2) and Simple Storage Service (S3) (AWS, 2011; AWS, 2006).

Authentication mechanisms may vary as in the case of WRMS. An example of VM provisioning in a cloud DIMS is illustrated in Fig. 2 for Eucalyptus Community Cloud (Nurmi, 2009).

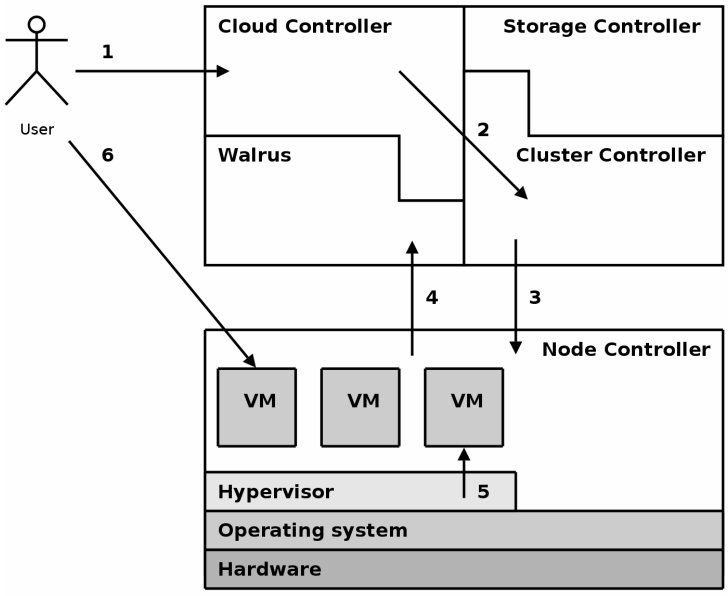

Figure 2. VM deployment with Eucalyptus Community Cloud.

The main steps for the deployment of a VM are: (1) a user requests a new VM to the Cloud Controller (CLC), (2) the CLC authorizes the request and forwards it to the Cluster Controller (CC), (3) the CC performs virtual network set-up (hardware and IP addresses, and firewall rules) and schedules the request to a Node Controller (NC), (4) the NC retrieves image files from Walrus (or cache), (5) the NC starts the VM through the hypervisor, and (6) the user logs into the VM.

Quite interestingly, Mateescu (2011) recognizes that both WRMS and DIMS can be well described by the managed computation factory model introduced by Foster (2006), and shown in Fig. 3. In this model, clients of computational resources do not directly interact with the bare resources; rather they work with the managed computation abstraction. A managed computation

can be (1) started, (2) stopped, (3) terminated, (4) monitored, and/or (5) controlled by interfacing with a managed computation factory. The factory is in charge of creating the managed computation, which is assigned to one or more computational resources. The managed computation often operates as a service; hence the name managed computation service in Fig. 3.

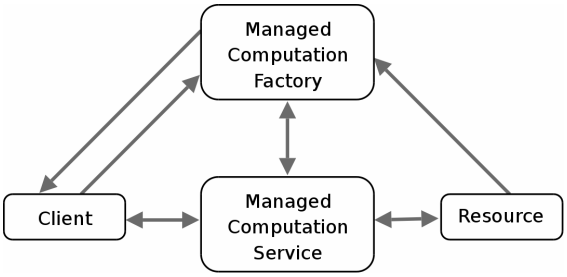

Figure 3. Managed Computation Factory model.

INTEGRATED PROVISIONING OF HETEROGENEOUS RESOURCES

In the previous sections we have discussed the opportunity of exposing scientific applications as web services, coordinated by workflows or business processes, all of them running on a "smart" aggregation of resources.

Resources are needed to run both web services and business processes, which may be well characterized in terms of the managed computation abstraction discussed in the previous section. Even if there is no fundamental difference between managed computations involving web services and business processes, we prefer to distinguish them at the logical level due to different responsibilities:

- web services exposing scientific applications or part of them have the sole responsibility of the computation on the assigned resources;
- business processes orchestrating one or more of the above web services are also in charge of (1) obtaining the resources needed for each computation represented by a web service and (2) managing their life cycle.

As a consequence, we distinguish two kinds of *resource requests*:

- *business service requests* for managed computations involving the execution of scientific applications, possibly exposed as web services or business services; and
- *workflow engine requests* for managed computations based on workflow enactment, which may as well expose workflow instances through a web service interface, i.e. a business process.

Resources, which can be physical (PM) or virtual (VM), are classified accordingly: worker nodes run scientific applications, while workflow engine nodes are dedicated to the execution of workflow engines (WfE). However, it should be noted that, at different times, a resource may act as a worker node or as a workflow engine node, depending on the managed computation assigned to it. In addition, all the resources belong to one or more WRMS and/or DIMS previously described, with the possible exception of some personal and specialized resource.

The model

To promote the integration, cooperation and inter-operation of heterogeneous resources from multiple providers as a cohesive aggregate, in such a way that they can be requested,

provisioned, used, monitored, and disposed without worrying about the underlying technological infrastructure, we propose the dynamic on-demand resource provisioning model depicted in Fig. 4. The model builds upon HPC, grid and cloud systems leveraging existing WRMS and DIMS, and connecting the different components through SOA interfaces whenever possible, directly or by means of suitable wrappers/adapters.

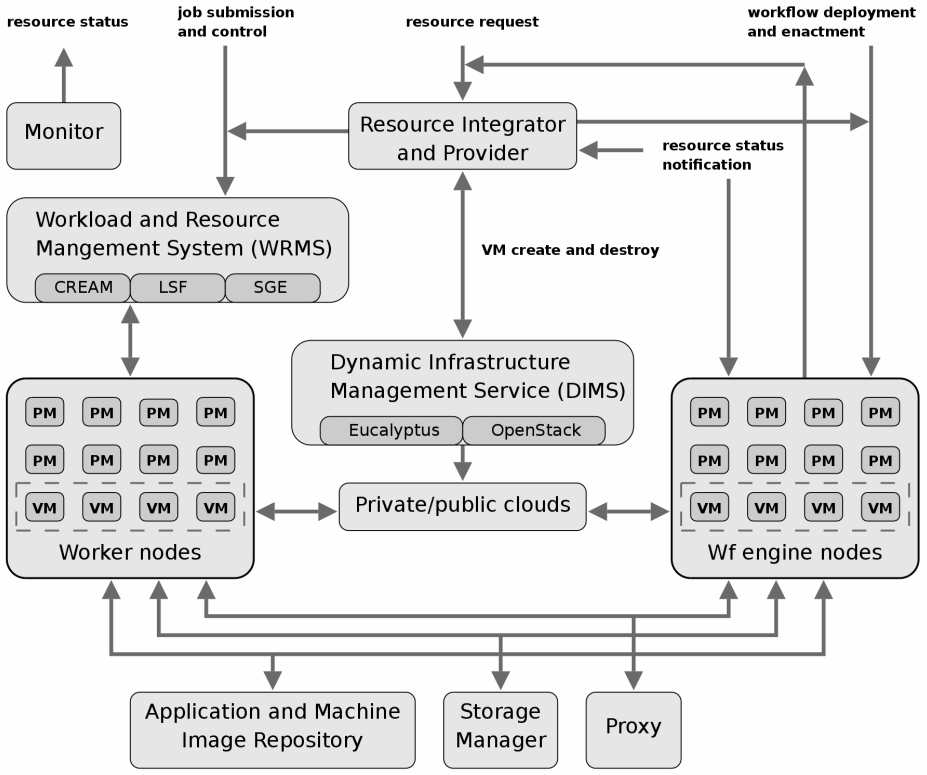

Figure 4. Model of dynamic on-demand resource provisioning system.

The glue of the model is constituted by the *Resource Integrator and Provider (ResIP)* which acts as a high level managed computation factory exposing a web service interface:

- it accepts requests for resources from clients;
- it translates and forwards requests to the underlying WRMS and DIMS;
- it receives resource heartbeats and notifications; and
- it notifies asynchronous clients of resource endpoint, availability and status changes.

Integrating WRMS and DIMS requires (1) that they expose a network-enabled public interface, possibly based on SOA standards, and (2) embedding the necessary logic into a module of the ResIP. Such integration is usually performed for entire classes of WRMS or DIMS: a specific instance may then provide its resources at any time as soon as a resource request refers to it. This allows, for example, drawing resources from two or more Eucalyptus Community Cloud infrastructures at the same time.

All resource requests are submitted to the ResIP, which takes the appropriate actions based on the request type, as we are going to describe in the following subsections together with the other components of the model. A successful resource request will start the desired managed

computation and provide the user with the computation service network endpoint and identifier: due to the nature of the resources and of their management systems this may not immediately happen, leaving users waiting for a variable amount of time. Even if this is not necessarily a problem, the ResIP offers different strategies to inform users when their managed computation is at last started; in particular, clients may submit their resource requests in a number of ways:

- a synchronous request returns successfully within a given timeout only if the computation starts before the timeout expires;
- an asynchronous request without notification returns immediately: clients may periodically poll the ResIP to know if the computation started; and
- an asynchronous request with notification returns immediately but allows clients to be notified as soon as the computation starts (clients must be able to receive notifications).

The ResIP is also responsible for disposing resources (canceling jobs in a WRMS, or terminating VMs in a DIMS) upon explicit request, resource lifetime expiration or in case of problems. The associated business service or workflow engine is informed of the imminent disposal, is explicitly in charge of its own clean-up (and should be implemented accordingly), and is required to notify the ResIP immediately before exiting.

User interaction

User interaction should be kept as much as possible (1) well-defined, (2) simple and, (3) independent of the WRMS/DIMS used, i.e. technology agnostic; conforming to SOA, user interaction is based on the exchange of XML documents.

Users wishing to run a workflow-based application should provide the following information:

- URL of the image of the workflow to be deployed and enacted on-demand (from a public repository or user-provided);
- URL of the images of all business services to be started on-demand and used by the workflow (from a public repository or user-provided); and
- endpoint and authentication credentials of the WRMS/DIMS required to provide the resources to run the on-demand workflow engine and business services.

Based on the above information, users (1) prepare and send to the ResIP the workflow engine request document (2), when the engine becomes ready, enact one or more workflow instances by preparing and sending the corresponding input documents and (3), when finished, send to the ResIP the release request for the engine.

According to the information specified in the input document, the workflow is in charge of (1) composing and sending to the ResIP the business service request documents for all on-demand services (2), when all business services are ready, invoking them in the specified order with the required input and (3), in the end, sending to the ResIP the release request for all business services.

The SoapUI application (SOAPUI, 2011) provides a GUI that can be of valuable help; after reading the WSDL document associated to a web service, it automatically lays out the structure of the XML input document leaving the user with the simpler task of filling in the data. SoapUI can also send the document to the given web service or workflow.

Workflow engine requests

A request for a resource dedicated to the execution of a workflow engine, workflow deployment, and the subsequent enactment of workflow instances, can be satisfied by materializing an appropriate VM in a cloud. The request specifies:

- class of cloud DIMS that will provide the resource, to determine the interface used for connecting;
- workflow engine type, needed for specific workflow deployment;
- workflow image, i.e. the URL of the archive containing the BPEL documents and related files;
- cloud DIMS requirements such as cloud provider, its interface endpoint, name of the VM image embedding the desired workflow engine, VM characteristics or flavor (number of CPUs, RAM size, etc.) and other cloud-specific parameters; and
- cloud authentication credentials.

Fig. 5 shows a fragment of the XML document representing a workflow engine request, where the cloud interface is Amazon EC2, the workflow engine is Apache Orchestration Director Engine (ODE, 2011), the workflow image can be downloaded from the URL http://ws-cyb.dsf.unica.it/ws/ode/DMProcesses1_macro.zip, the cloud provider is Eucalyptus with its own set of requirements (endpoint, region, zone, VM image and flavor) and credentials (username/password).

```
<resourceRequest>
  <BSRequest>
    <BSScheduler>EC2</BSScheduler>
    <BSType>ODE</BSType>
    <BSURL>http://ws-cyb.dsf.unica.it/ws/ode/DMProcesses1_macro.zip</BSURL>
    <BSRequirements>
      <EC2Params>
        <ec2Provider>eucalyptus</ec2Provider>
        <ec2Endpoint>http://149.165.146.135:8773/services/Eucalyptus</ec2Endpoint>
        <ec2Region>Eucalyptus</ec2Region>
        <ec2Zone>indiafg</ec2Zone>
        <ec2Image>emi-54090E06</ec2Image>
        <ec2Flavor>c1.medium</ec2Flavor>
      </EC2Params>
    </BSRequirements>
    <BSCredentials>
      <CryptedPWAuth>
        <username>...</username>
        <cryptedPW>...</cryptedPW>
      </CryptedPWAuth>
    </BSCredentials>
  </BSRequest>
</resourceRequest>
```

Figure 5. XML fragment representing a workflow engine request.

When the ResIP processes the request, a VM creation command is submitted to the cloud DIMS. The latter schedules the instantiation of a new VM of the given flavor on its pool of PM, and returns a VM identifier or an error. As soon as the VM is up and running, the embedded workflow engine is started, and the workflow image is downloaded and deployed; the ResIP is then notified of the WfE interface endpoint.

As soon as the deployment is completed, the user can enact the workflow by providing the desired input document; many workflow instances may be enacted within the same WfE with different input documents. When the WfE is no longer needed, the user sends a resource release request to the ResIP which disposes the corresponding resource by sending a VM destroy command to the DIMS.

A workflow is an abstract representation of a scientific application and is made of a set of activities or jobs with precedence constraints between them (Singh, 2006). Each job needs a resource to run on, and the workflow places the corresponding resource requests on the ResIP as described in the next subsection.

Business service requests

A request for a resource needed to run a job, i.e. a module of a scientific application in the form of a business service, specifies:

- notification endpoint, used by the business service to advertise itself once ready, and usually pointing to the client that submitted the request (e.g. a workflow);
- class of HPC/grid WRMS or cloud DIMS that will provide the resource, to determine the interface used for connecting;
- web service (application) image, i.e. the URL of the archive containing the web service executables and related files;
- WRMS or DIMS requirements such as interface endpoint and other specific parameters; and
- authentication credentials.

Fig. 6 shows a fragment of the XML document representing a business service request for a grid resource, where the WRMS is gLite CREAM, the WS image can be download from the URL http://ws-cyb.dsf.unica.it/ws/jar/PrimeNumber.jar, the virtual organization is cybersar, the LRMS batch system is Platform Load Sharing Facility (LSF, 2005), the batch queue is cybersar, and the authentication is performed through an X509 proxy certificate. After start-up the business service will advertise itself to the client endpoint http://10.0.0.5:8099/ode/processes/DMWrapperProcess1_RANotification.

```
<resourceRequest>
  <notificationEndpoint>
    <wsa:EndpointReference
     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
      <wsa:Address>
        http://10.0.0.5:8099/ode/processes/DMWrapperProcess1_RANotification
      </wsa:Address>
      <wsa:ServiceName portName="ResourceAllocatorNotificationPort"
       xmlns:ra="http://wsdl.resources.cybersar">
        ra:DMWrapperProcess1_RANotification
      </wsa:ServiceName>
    </wsa:EndpointReference>
  </notificationEndpoint>
  <BSRequest>
    <BSScheduler>CREAM</BSScheduler>
    <BSURL>http://ws-cyb.dsf.unica.it/ws/jar/PrimeNumber.jar</BSURL>
    <BSRequirements>
      <CREAMParams>
        <creamEndpoint>https://cecream-cyb.ca.infn.it:8443</creamEndpoint>
        <creamVO>cybersar</creamVO>
        <creamBatchSystem>lsf</creamBatchSystem>
        <creamQueue>cybersar</creamQueue>
      </CREAMParams>
    </BSRequirements>
    <BSCredentials>
      <ProxyAuth>
        <proxy>...</proxy>
      </ProxyAuth>
    </BSCredentials>
  </BSRequest>
</resourceRequest>
```

Figure 6. XML fragment representing a business service request.

The ResIP forwards the requests to the specified DIMS or WRMS. A conventional job submitted to a HPC or grid batch system contains the instructions needed to start the computation, written according to some JDL. In our case, the job responsibility is to start the web service (as a sub-process) and wait until it stops, while the real computation is orchestrated by a workflow through one or more invocations of the web service. The ResIP is in charge of preparing the job submitted to the WRMS by translating the specifications contained in the business service request document into the corresponding JDL. Job submission to a WRMS returns a job identifier or an error. Fig. 7 shows the JDL file generated by translating the business service request document represented in Fig. 6.

```
[
  VirtualOrganisation      = "cybersar";
  Executable               = "runbs.135";
  Arguments                = "";
  BatchSystem              = "lsf";
  QueueName                = "cybersar";
  StdOutput                = "std.out";
  StdError                 = "std.err";
  InputSandbox             = "runbs.135";
  InputSandboxBaseDestUri  = "gsiftp://localhost";
  OutputSandbox            = {"std.out", "std.err"};
  OutputSandboxBaseDestUri = "gsiftp://localhost";
]
```

Figure 7. JDL file generated from the business service request document of Fig. 6.

If the job is to be executed by a VM allocated on a cloud, the VM is created as described in the previous subsection, except that a standard machine image is used unless a custom image is specified; the VM is instructed to execute the job after booting the operating system.

As soon as the job is started on a worker node, be it a PM or VM, it notifies the workflow that has placed the request (via the ResIP) of the endpoint of the web service, which is ready and listening for incoming messages. When all the resources requested by the workflow are available, the execution can proceed with the invocation of all the required web services in the specified order.

External resources

In the foreseen environment, specialized nodes on the network may run static services that can be invoked from within a workflow. In addition, an interesting degree of interactivity may be added to plain workflow execution by invoking a visualization service, e.g. at the end of each iteration in an iterative simulation. Such visualization service may well be executed on a personal resource, such as the user desktop, to give a real-time visual feedback of the state of the simulation.

If the service endpoints are “immutable”, they can simply be hard-coded into the workflow description, but the visualization service presents a problem. Hard-coding a user-specific endpoint (the desktop network address) makes the workflow unusable by different users. We then need to account for resource requests that refer to external resources and for which creation is not needed. This allows workflows to manage all jobs exactly in the same way, delegating to the ResIP the responsibility for provisioning a new resource only if needed. To this end, business service requests needing a personal resource are handled in the following way: (1) the request is submitted with WRMS/DIMS class set to “MANUAL” and the notification endpoint set to point to the workflow, (2) the user starts the business service on the personal resource, (3) after start-up the business service notifies the workflow of its endpoint, (4) the workflow invokes the business service at the provided dynamic endpoint.

Proxy

In general, worker nodes and workflow engine nodes live on different private networks which are not directly connected to one another (the only assumption we do is that they can open network connections towards the Internet through some kind of Network Address Translation or

NAT service), so a *Proxy* service is essential in routing messages from business processes (workflows) to business services (web services) and vice-versa.

Other components

The other components that complete the model are briefly described here, leaving some details to next section. The *Storage Manager* provides temporary storage to data-intensive business services. The *Monitor* is a simple service that can be queried for resource status information. The *Application and Machine Image Repository* hosts business service, workflow and virtual machine images that will be executed on the provisioned resources.

IMPLEMENTATION

In this section we describe a working implementation of the model previously outlined. According to the general philosophy of a SOA-based framework, we have developed a number of infrastructure (web) services corresponding to various components of the model - using the Java programming language - and re-used open-source tools and systems whenever possible, operating the necessary integration. Fig. 8 shows the component diagram of the presented implementation with the basic dependencies between components, as described in the next subsections.

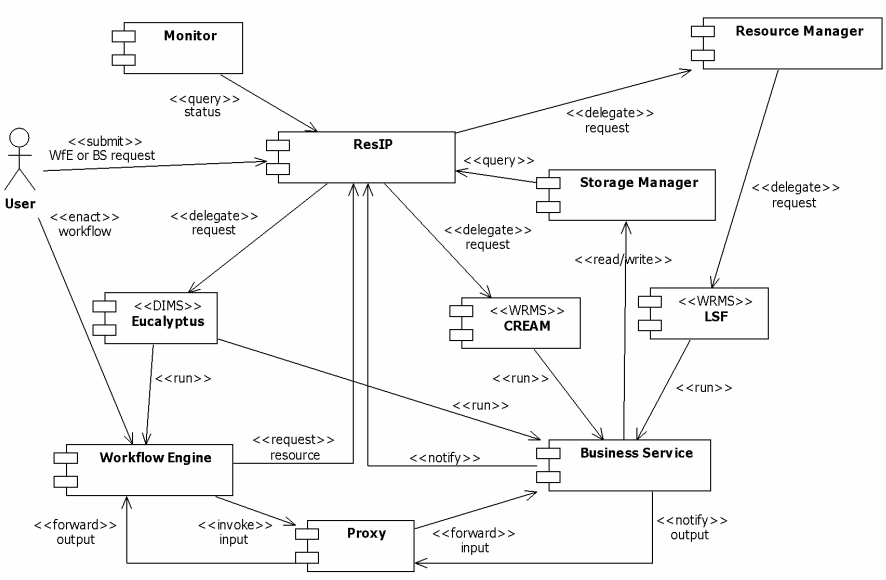

Figure 8. Component diagram with basic dependencies.

Integration of WRMS and DIMS

As already discussed, integrating WRMS and DIMS requires that they expose a network-enabled public interface, possibly based on SOA standards. In this work, we have developed ResIP modules to interact with the following WRMS:

- gLite v3.2 Compute Element (CE);
- gLite v3.2 Computing Resource Execution And Management (CREAM);
- Platform Load Sharing Facility v6.2 (LSF); and
- Oracle Grid Engine v6.2 (SGE, 2009).

CE and CREAM expose OGSA-compliant interfaces and, in addition, it is possible to resort to existing Java API such as jLite (Sukhoroslov, 2009). On the contrary, LSF does not offer a similar functionality, so we have developed a simple but effective web-service wrapper, the *Resource Manager*, and deployed it to one of the submission hosts, on top of the LSF command line interface. We have followed the same strategy for SGE, too, even if the latter is supported by DRMAA.

On the cloud side, we have integrated:

- OpenNebula, defined by its developers an "industry standard open source cloud computing tool" (Sotomayor, 2009);
- OpenStack, an open-source porting of RackSpace's DIMS (Pepple, 2011); and
- Eucalyptus Community Cloud, presented as "a framework that uses computational and storage infrastructure commonly available to academic research groups to provide a platform that is modular and open to experimental instrumentation and study" (Nurmi, 2009).

According to their documentation, all these systems expose a RESTful interface compatible with Amazon EC2 and S3 API, which are becoming a de-facto standard. In practice the compatibility is not full, but for OpenStack and Eucalyptus we have successfully employed such interface with the help of the jclouds library (JCLOUDS, 2011). OpenNebula needs additional components to expose the EC2 interface, so we have decided to use its native interface, which is anyway satisfactory.

Resource Integrator and Provider

In our implementation the Resource Integrator and Provider functionality is mapped onto a hierarchy of Java classes and is exposed through a web service interface. It would be impossible to describe all classes in detail here, so we only list their key responsibilities:

- process all resource requests and make up the corresponding jobs;
- interact with WRMS and DIMS for job and VM scheduling, execution and control;
- receive status notifications from resources and deliver them to the interested entities (e.g. workflows or monitoring applications);
- dispose resources;
- manage resource context and status;
- register proxies, storage managers and monitoring applications;
- trace message flow and inform registered monitoring applications;
- enforce elementary security practices: authentication credentials containing plain text passwords must be encrypted with the ResIP public key before transmission; if necessary, the use of one-time security tokens can be enabled; and
- assign registered proxies and storage managers to business services.

The Java classes implementing the ResIP functionality work with an abstract representation of business service resources, embodied by the class BSResource, which does not depend on the resource type. The dynamic behavior of an abstract resource is captured by the state diagram shown in Fig. 9.

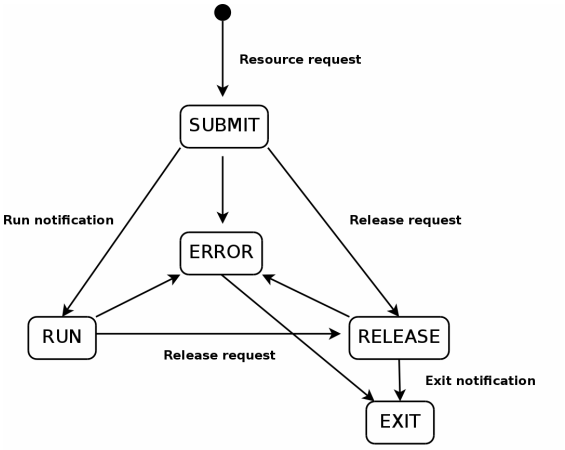

Figure 9. State diagram for an abstract resource.

If a resource request is permissible, a new instance of the specific BSResource sub-class is created with status set to SUBMIT and the corresponding physical or virtual resource creation advances as previously described. As soon as the resource becomes ready, the associated business service sends a (resource status) notification message to the ResIP and such event triggers a change of status from SUMBIT to RUN. Only at this point the resource can be used and the ResIP notifies all the interested entities. Once the resource is no longer needed, it may be disposed; its status is then set to RELEASE and the disposal of the corresponding physical or virtual resource proceeds. The business service sends another (resource status) notification message that causes the status to change from RELEASE to EXIT.

Network connectivity

Network connectivity between the different components is guaranteed by the following assumptions:

- the ResIP must live on a public network so that every other component can contact it, including all the users of the system which, in a collaborative distributed environment, can be located anywhere;
- worker and WfE nodes, as already observed, may live on different private networks which are not directly connected to one another, but we assume that they can open network connections towards the Internet through some kind of NAT service. The same is true for personal resources;
- WRMS and DIMS, if not living on a public network, must be accessible from the ResIP (e.g. via a virtual private network or VPN);
- the Application and Machine Image Repository may be on a public network or may be replicated on the private networks attached to each pool of PM/VM; and
- the Storage Manager must live on a public network to allow for the sharing of data.

In addition the ResIP (public network) is required to notify a WfE (private network) when a requested business service resource is ready and this can be achieved by setting up a VPN between the WfE node and the ResIP: the VM that executes the WfE can be instructed to initialize the VPN during start-up. The Proxy service described in the next subsection completes the picture as previously discussed. The resulting network connectivity is synthesized in Fig. 10.

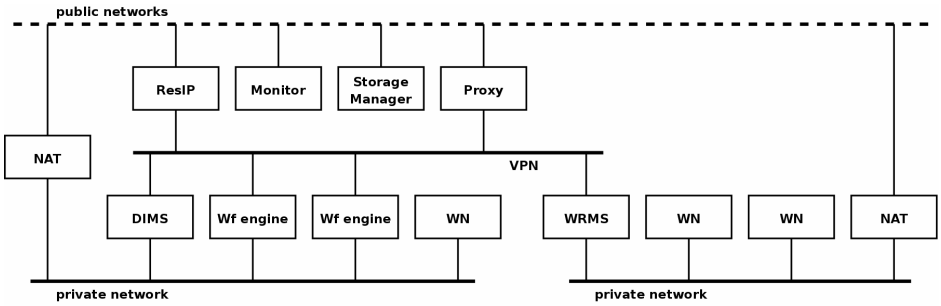

Figure 10. Schema of network connectivity.

Proxy

The Proxy service acts as an intermediary for routing (SOAP) messages from WfE to business services and vice-versa. When the ResIP needs to notify a WfE about the endpoint (URL) of a newly started business service, it replaces the business service private network address with the Proxy public address. In this way, the messages sent by a WfE to a business service are in fact sent to the Proxy, which performs a sort of message routing to its final destination, based on a special message tag. Every business service, upon start-up, opens a persistent bi-directional network connection with the Proxy; the latter will use this channel to route all the messages directed to the service.

Workflow engines

As suggested by Bosin (2011a) and Dörnemann (2009), the choice of a workflow language naturally leads to the Business Process Execution Language (OASIS, 2007) which belongs to the family of SOA standards. BPEL is complementary to the web service definition language (W3C, 2001) and in fact BPEL instances are themselves web services. Both languages are XML-based and both make use of the XML schema definition language. A number of BPEL engines were considered:

- Sun BPEL engine for Glassfish server;
- Oracle BPEL Process Manager in the Oracle SOA Suite 11g (ORACLE, 2011);
- ActiveVOS BPEL execution engine (AVOS, 2011);
- Orchestra (OW2, 2011); and
- Apache Orchestration Director Engine (ODE).

Due to our preference for open-source software and self-contained tools, we have chosen to work with ODE v1.3.5, leaving the integration of other engines for the future. This requires (1) a module in the ResIP to manage WfE requests, (2) a simple SOA wrapper to interface ODE with the ResIP and clients, and (3) a VM image embedding ODE.

Business services

Business services implement the business logic of scientific applications, usually in a modular form where the whole logic is partitioned into re-usable, loosely-coupled, cohesive software modules. In a SOA-based framework, business services expose their functionality through a web service interface formally described by a WSDL document. To be integrated in our model,

business services, in addition to the core scientific functionality just described, must implement some housekeeping operations:

- immediately after start-up they must notify the ResIP of their status (RUN);
- immediately before exiting they must notify the ResIP of their status (EXIT);
- they must retrieve from the ResIP the assigned Proxy and Storage Manager service endpoints;
- they should expose some standard operations such as version(), load(), exit(), etc.; and
- they should provide asynchronous operations for long-running computations.

Other services

As already discussed, Storage Managers provide temporary storage to data-intensive business services. When the output from a business service serves as input for a subsequent service invocation during workflow execution, it would be ineffective to transfer the data back to the workflow and forth again to the next service. The Storage Manager handles plain and efficient HTTP uploads and downloads, assigning a unique public URL to the uploaded data so that they can be easily retrieved later when needed.

The Monitor is a simple service that can be queried for information about the resource context related to business services, such as resource identifier, status, WSDL port type, last operation invoked and the related time-stamp.

The Application and Machine Image Repository is distributed over a number of components: a simple HTTP server hosts all the business service and workflow images, while every DIMS manages its own dedicated virtual machine image repository.

AUTHENTICATION AND SECURITY

In the current implementation, resource allocation is done using real-life authentication mechanisms such as grid proxy certificates or digital signatures, but the access to scientific services is granted by using the resource identifiers provided to users by the ResIP and contained in the input messages. In the general case, where stronger security is needed, an authorization token (e.g., a session password) or the X.509 authentication framework with the Web Services Security could be used together with HTTPS protocol to transport SOAP messages (as with Grid Services). In a prototype environment this is an additional complication, and we have not yet implemented it.

While authorization is performed directly by WRMS/DIMS (e.g. LCAS with gLite or access control lists in OpenNebula), here we give some details related to user authentication.

- gLite authentication: a VOMS proxy certificate for a given VO is manually generated by the grid user through gLite or jLite command line interfaces and included in the resource request. In this case, users are required to have (1) valid grid credentials for the VO and (2) access to a gLite user interface to be able to issue the proxy generation commands. The ResIP employs the provided VOMS proxy to perform user authentication with gLite CE or CREAM.
- LSF and SGE authentication: an LDAP server provides password-based user authentication, but this is not suitable over a network, and we have added public key certificates for our users to the LDAP server. User authentication is then performed by verifying the digital signature placed on an authentication token provided by the ResIP.

- Cloud authentication: Eucalyptus Community Cloud, OpenNebula and OpenStack provide basic password-based user authentication; in addition OpenStack and OpenNebula can be configured to use public key authentication.

To prevent clear text passwords from floating around in XML files, which may be exchanged among users, the latter are forced to encrypt passwords with ResIP public key (together with a one-time token), by means of a simple command line tool which outputs the proper XML fragment (i.e. the element `<CryptedPWAuth>` in Fig. 5).

RESULTS AND DISCUSSION

In this section we wish to discuss some results and make some considerations about:

- simple performance measurements;
- practical problems and possible solutions; and
- limitations of the model.

The performance measurements refer to a case study focusing on a specific bio-informatics domain: the application of machine learning techniques to molecular biology. In particular, micro-array data analysis (Bosin, 2007) is a challenging area since it has to face with data-sets composed by a relatively low number of records and characterized by an extremely high dimensionality.

The workflow developed for this case study is a nested workflow. The invoked process, or *core process*, orchestrates a single iterative computation with multiple invocations of the data mining operations exposed by a business service. The latter can download the needed data files from a web repository and reads/writes the data generated during the computation from/to the Storage Manager. The invoking process, or *wrapper process*, manages the concurrent execution of a variable number of core processes. Optionally, the workflow can link to an external visualization service, running on the user desktop, for monitoring the execution of these processes.

Performance measurements

To measure the time needed to start a managed computation we have considered three main phases; (1) the request processing time is small compared to other times; (2) the resource set-up time depends on WRMS/DIMS and is similar for the employed HPC and grid systems (on average about 1 minute) and higher for cloud systems, unless special tuning is performed as discussed by Bosin (2012) (on average 5-10 minutes which can be reduced to 1 minute); (3) workflow engine and business service start-up times are comparable to resource start-up times and depend on the speed of the network for downloading images.

VM performances might be an issue, so we have compared execution times using PM and VM both for workflow engines and business services and found no substantial differences; our tests require relevant CPU and network I/O activities, and no heavy local disk I/O which may be slow on VM (Bosin, 2012).

The workflow described above has been enacted with the following parameters: the wrapper process starts 10 concurrent instances of the core process, each invoking its own business service during 20 iterations; business services have been executing both on homogeneous resources and on a mix of geographically distributed heterogeneous infrastructures such as Cybersar (2006) and FutureGrid (2009). The generated data flow is the following: 2000 SOAP messages exchanged through the Proxy service, 600 data files downloaded from web repositories (approx. 4.5 GB of

data), 800 write operations and 1000 read operations to/from Storage Manage service. The execution time varies with the resource hardware but also with the available network bandwidth between business services, data repositories and Storage Manager.

Many concurrent enactments of the workflow can be used to assess system scalability: with the available resources, 10 concurrent workflows were enacted (i.e. 10 workflow engines on cloud and 100 business services half on cloud and half on HPC/grid) without any particular problem except for some transient failures in providing the requested resources occurred with some WRMS/DIMS (mostly Eucalyptus). From the tests performed we expect that the system can manage a number of concurrent business services at least up to 1000 if the Proxy service is replicated as discussed later in this section.

A key point in workflow engine and overall system scalability and performance is the management of the data flow for data intensive computations. Many workflow engines (including BPEL) act as brokers for all message exchanges between the business services participating in the workflow, but embedding large data-sets into such messages is not efficient and can lead to a collapse of both the engine and the Proxy service. Rather, web repositories and the Storage Manager should be used, and data-sets can be indirectly passed to business services as URL references; in this way, each business service can directly and efficiently read/write data from/to repositories and the Storage Manager thus minimizing transfers and improving scalability and performance.

Problems and solutions

A number of problems are related to the BPEL language and the engine implementations. The BPEL specifications do not allow multiple receive activities in the same scope with the same port type and operation. This is an annoying limitation which may be overcome by duplicating the operations with different names. Different BPEL engines miss some of the functionality dictated by the standard, or implement non standard features. An example is variable initialization, required by some engines and automatically performed by others; another is *partnerLink* assignment whose syntax can be different depending on the engine. In ODE, web service functionality is built on top of Apache Axis2 (AXIS2, 2011), the older Apache WS framework, and many advanced features are missing, for example WS-Security.

Another point that deserves some attention is the disposal of resources when they are no longer needed. In principle, the resource life cycle, i.e. provisioning, usage and disposal, can be managed through the workflow if the correct support is provided at the business service level. If no error occurs things go as expected, but if only the workflow fails before disposal, a bunch of resources may be left around reserved, unused and in some cases charged for. In addition, a failure with a resource or business service may leave a workflow indefinitely waiting for it; and the list of possible errors is long. BPEL fault and compensation handlers may help in designing better error resilience but can do nothing if the workflow engine or its VM crashes. The infrastructure services (ResIP, DIMS and WRMS) must then be charged of the extra responsibility of resource disposal when something goes wrong, e.g. by periodically checking resource status for failures or long inactivity periods or other abnormal behavior.

Limitations

The model presented in this work represents an attempt aimed at experimenting possible ways to cope with the general and complex problem of the integration of heterogeneous resources from

multiple providers as a cohesive aggregate, and unavoidably has a number of limitations and aspects not (adequately) covered; among others:

- Scalability with the number of workflow instances and business services can be an issue for the Proxy service since it mediates most message exchanges; to account for this more than one proxy can register with the ResIP, which assigns one of the available proxies to each business process and service thus ensuring proper load balancing. ResIP scalability has not been assessed except for the consideration that ResIP and all related services can be replicated allowing for static load balancing (the same WRMS/DIMS can be shared by different ResIP). The Storage Manager service can be replicated, too, but assignment to a business process or service should be based on some kind of network distance metric.
- Redundancy of the ResIP service could be arguably obtained by means of standard high availability techniques (LHA, 2011) if persistent data is stored into independent and replicated DBMS; in addition, the use of multiple Proxy and Storage Manager services, as suggested above, should also help to guarantee the necessary redundancy.
- Application recovery in case of network or other failures has not been taken into consideration; the simple solution of automatically re-enacting failed workflows may work in case of transient failures but will not work in case of structural failures (e.g. when a computation exceeds the memory limit of a resource).
- The need for a meta-scheduler has not been investigated, since WRMS and DIMS provide their own schedulers; if needed, such a meta-scheduler should be able to manage resource requests with very different requirements and resource pools (WRMS and DIMS) that change over time and over users.

Another important point concerns existing applications, domain-specific and resource-specific tools; many VO have developed their own domain-specific, workflow scheduling systems which provide extensive monitoring, scheduling, data-access and replication and user priority management capabilities. We expect that users may be concerned by the problems/efforts required to use or even re-implement such applications and tools in a new environment (Gentzsch, 2009). According to what we said in the introduction, however, our proposal is not meant to “replace” but rather to “complement”, so we must be pragmatic: it makes no sense to bother users which are happy with their applications, tools and resources. Many “conventional” users will simply have no benefits from dynamic on-demand provisioning of resources, but many others may be attracted by this on one side and by the advantages offered by the SOA approach on the other, and may be interested in creating new or better applications using existing ones as building blocks; in most cases it will not be necessary to re-implement applications and tools but simply wrap them with a SOA interface.

CONCLUSION

In this paper we have presented a model for the dynamic on-demand provisioning of computational resources from multiple providers, based on the convergence and integration of different computing paradigms and infrastructures. Heterogeneous distributed resources are presented to their users with the illusion of a cohesive aggregate of resources, accessible through a well-defined standards-based software interface.

In our opinion, the model benefits from a number of qualifying and distinctive features; it (1) is founded on firm SOA principles; (2) integrates different computing paradigms, systems and infrastructures; (3) is open to further integrations and extensions both in terms of new resource management systems (WRMS and DIMS) and workflow management systems; (4) takes

advantage from the integration and re-use of open-source tools and systems; (5) some of the components can be easily made scalable and redundant; and (6) promotes the development of new applications exposed as web services and managed by workflows.

As opposed to many other systems proposed in the literature, which focus on a single paradigm, this work is a first attempt aimed at complementing, integrating and building on existing HPC, grid and cloud paradigms, by playing the role of a dynamic resource aggregator exposing a technology agnostic abstraction layer. The proposed approach is quite general and flexible. It has been applied to bioinformatics environments and has returned a set of results and feedbacks regarding its implementation, usage, and the overall feasibility. However, application scenarios are not limited and span brain dynamics investigation, geo-informatics and drug discovery, etc.

Issues to be addressed arise in various areas, such as workflow execution monitoring, fault handling and compensation, scalability, policy enforcement, trust and security support, quality of service monitoring, transaction logging, and so on.

ACKNOWLEDGEMENTS

The author acknowledges the Cybersar Consortium for the use of its computing facilities. This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

REFERENCES

Aiftimiei C., Andreetto P., Bertocco S., Dalla Fina S., Dorigo A., Frizziero E., Gianelle A., Marzolla M., Mazzucato M., Sgaravatto M., Traldi S. & Zangrando L. (2010). Design and implementation of the gLite CREAM job management service, *Future Generation Computer Systems*, 26, 654-667.

Akram A., Meredith D. & Allan R. (2006). Evaluation of BPEL to scientific workflows. In *Proceedings of CCGRID'06* (pp. 269-274). Washington, DC, USA: IEEE Computer Society.

AVOS (2011). ActiveVOS platform. Retrieved Apr. 25, 2011, from http://www.activevos.com

AWS (2006). Amazon Web Services: Amazon Simple Storage Service - API Reference. Retrieved Apr. 24, 2011, from http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf

AWS (2011). Amazon Web Services: Amazon Elastic Compute Cloud - API Reference. Retrieved Apr. 24, 2011, from http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-api.pdf

AXIS2 (2011). Apache Axis2. Retrieved Jun. 5, 2011, from http://axis.apache.org/axis2/java/core

Bosin A., Dessì N. & Pes B. (2007). A Cost-Sensitive Approach to Feature Selection in Micro-Array Data Classification, *Lecture Notes in Computer Science, vol. 4578* (pp. 571-579). Berlin, Germany: Springer.

Bosin A., Dessì N. & Pes B. (2011a). Extending the SOA paradigm to e-Science Environments, *Future Generation Computer Systems*, 27, 20-31.

Bosin A., Dessì N., Bairappan M. & Pes B. (2011b). A SOA-Based Environment Supporting Collaborative Experiments in E-Science, *International Journal of Web Portals*, 3(3), 12-26.

Bosin A., Dessalvi M., Mereu G. M. & Serra G. (2012). Enhancing Eucalyptus Community Cloud, *Intelligent Information Management*, 3(4), 52-59.

Brandic I., Pllana S. & Benkner S. (2006). High-level composition of QoS-aware Grid workflows: an approach that considers location affinity. In: *Proceedings of WORKS06* (pp. 1-10). Paris, France.

Chao K., Younas M., Griffiths N., Awan I., Anane R., & Tsai C.-F. (2004). Analysis of Grid service composition with BPEL4WS. In: *Proceedings of AINA'04* (p. 284). Los Alamitos, CA, USA: IEEE Computer Society.

Churches D., Gombas G., Harrison A., Maassen J., Robinson C., Shields M., Taylor I. & Wang I. (2006). Programming scientific and distributed workflows with Triana services, *Concurrency and Computation: Practice and Experience*, 18 (10), 1021-1037.

Cybersar (2006). Cybersar consortium for supercomputing, computational modeling and management of large databases. Retrieved Jan. 22, 2012, from http://www.cybersar.com

Deelman E., Singh G., Su M., Blythe J., Gil Y., Kesselman C., et al. (2005). Pegasus: a framework for mapping complex scientific workflows onto distributed systems, *Scientific Programming*, 13 (3), 219-237.

Deelman E., Gannon D., Shields M. & Taylor I. (2009). Workflows and e-Science: an overview of workflow system features, *Future Generation Computer Systems*, 25, 528-540.

Dörnemann T., Friese T., Herdt S., Juhnke E. & Freisleben B. (2007). Grid workflow modeling using Grid-specific BPEL extensions. In: *Proceedings of German e-Science Conference*. Baden-Baden, Germany.

Dörnemann T., Juhnke E. & Freisleben B. (2009). On-demand resource provisioning for BPEL workflows using Amazon's elastic compute cloud. In F. Cappello, C. Wang, R. Buyya (eds.), *9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (pp. 140-147). Piscataway, NJ, USA: IEEE Computer Society.

EGEE (2010). Enabling Grids for E-sciencE project. Retrieved Apr. 11, 2011, from http://www.eu-egee.org

Elmroth E., Hernandez F. & Tordsson J. (2010). Three fundamental dimensions of scientific workflow interoperability: model of computation, language and execution environment, *Future Generation Computer Systems*, 26, 245-256.

Emmerich W., Butchart B., Chen L., Wassermann B. & Price S. (2005). Grid service orchestration using the business process execution language (BPEL), *Journal of Grid Computing*, 3 (3-4), 283-304.

Fahringer T., Prodan R., Duan R., Hofer J., Nadeem F., Nerieri F., et al. (2007). ASKALON: a development and Grid computing environment for scientific workflows. In: Taylor I., Deelman E., Gannon D., Shields M. (eds.), *Workflows for eScience: Scientific Workflow for Grids* (pp. 450-471). Berlin, Germany: Springer-Verlag.

Foster I., Kesselman K., Nick J. M. & Tuecke S. (2002). The Physiology of the Grid - An Open Grid Services Architecture for Distributed Systems Integration Globus Alliance. Retrieved Jan. 24, 2012, from http://www.globus.org/alliance/publications/papers/ogsa.pdf

Foster I., Keahey K., Kesselman C., Laure E., Livny M., Martin S., Rynge M. & Singh G. (2006). *Embedding community-specific resource managers in general-purpose grid infrastructure* (Tech. Rep. ANL/MCS-P1318-0106). Lemont, IL, USA: Argonne National Laboratory.

Fox G. & Gannon D. (2006). *A survey of the role and use of web services and service oriented architectures in scientific/technical Grids* (Tech. Rep. 08/2006). Bloomington, IN, USA: Indiana University.

FutureGrid (2009). FutureGrid: a distributed testbed for Clouds, Grids, and HPC. Retrieved Nov. 17, 2011, from https://portal.futuregrid.org

Gentzsch W. (2009). Porting Applications to Grids and Clouds, *International Journal of Grid and High Performance Computing*, 1(1), 55-77.

JCLOUDS (2011). Jclouds multi-cloud library. Retrieved Apr. 24, 2011, from http://code.google.com/p/jclouds

Laure E., Fisher S. M., Frohner A., Grandi C., Kunszt P., Krenek A., Mulmo O., Pacini F., Prelz F., White J., Barroso M., Buncic P., Hemmer F., Di Meglio A. & Edlund A. (2006). Programming the Grid with gLite, *Computational Methods in Science and Technology*, 12(1), 33-45.

Leymann F. (2006). Choreography for the Grid: towards fitting BPEL to the resource framework, *Concurrency and Computation: Practice and Experience*, 18 (10), 1201-1217.

LHA (2011). Linux-HA. Retrieved Jan. 13, 2012, from http://www.linux-ha.org/wiki/Main_Page

LSF (2005), Platform Load Sharing Facility. Retrieved Jun. 9, 2011, from http://www.platform.com/workload-management/high-performance-computing

Ma R. Y., Wu Y. W., Meng X. X., Liu S. J. & Pan L. (2008). Grid-enabled workflow management system based on BPEL, *International Journal of High Performance Computing Applications*, 22 (3), 238-249.

Mateescu G., Gentzsch W. & Ribbens C. J. (2011). Hybrid Computing – Where HPC meets grid and Cloud Computing, *Future Generation Computer Systems*, 27, 440-453.

McPhillips T., Bowers S., Zinn D. & Ludascher B. (2009). Scientific workflows for mere mortals, *Future Generation Computer Systems*, 25, 541-551.

Mietzner R. & Leymann F. (2008). Towards provisioning the cloud: on the usage of multigranularity flows and services to realize a unified provisioning infrastructure for SaaS applications. In *Proceedings of IEEE Congress on Services* (pp. 3-10). Los Alamitos, CA, USA: IEEE Computer Society.

Murphy M. A. & Goasguen S. (2010). Virtual Organization Clusters: Self-provisioned clouds on the grid, *Future Generation Computer Systems*, 26, 1271-1281.

Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L. & Zagorodnov D. (2009). The Eucalyptus Open-Source Cloud-Computing System. In F. Cappello, C. Wang, R. Buyya (eds.), *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (pp. 124-131). Piscataway, NJ, USA: IEEE Computer Society.

OASIS (2007). Web Services Business Process Execution Language Version 2.0. Retrieved Jun. 9, 2011, from http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

ODE (2011). Apache Orchestration Director Engine. Retrieved Jun. 9, 2011. from http://ode.apache.org

Oinn T., Addis M., Ferris J., Marvin D., Senger M., Greenwood M., et al. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics*, 20 (17), 3045-3054.

Oinn T., Li P., Kell D. B., Goble C., Goderis A., Greenwood M., Hull D., et al. (2007). Taverna/myGrid: aligning a workflow system with the life sciences community. In: Taylor I., Deelman E., Gannon D., Shields M. (eds.), *Workflows for eScience: Scientific Workflow for Grids* (pp. 300-319). Berlin, Germany: Springer-Verlag.

ORACLE (2011). Oracle BPEL Process Manager. Retrieved Jun. 9, 2011, from http://www.oracle.com/technetwork/middleware/bpel/overview/index.html

OW2 (2011). Orchestra User Guide. Retrieved Jan. 27, 2012, from http://download.forge.objectweb.org/orchestra/Orchestra-4.9.0-UserGuide.pdf

Pennington D. D., Higgins D., Townsend Peterson A., Jones M. B., Ludäscher B. & Bowers S. (2007). Ecological Niche modeling using the Kepler workflow system. In: Taylor I., Deelman E.,

Gannon D., Shields M. (eds.), *Workflows for eScience: Scientific Workflow for Grids* (pp. 91-108). Berlin, Germany: Springer-Verlag.

Pepple K. (2011). *Deploying OpenStack.* Sebastopol, CA, USA: O'Reilly Media.

SGE (2009). Oracle Grid Engine. Retrieved Jun. 9, 2011, from http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html

Singh G., Kesselman C. & Deelman E. (2006). Application-level resource provisioning on the grid. In *IEEE International Conference on e-Science and Grid Computing* (p. 83). Piscataway, NJ, USA: IEEE Computer Society.

Slominski A. (2007). Adapting BPEL to scientific workflows. In: Taylor I., Deelman E., Gannon D., Shields M. (eds.), *Workflows for eScience: Scientific Workflow for Grids* (pp. 208-226). Berlin, Germany: Springer-Verlag.

SOAPUI (2011). Eviware SoapUI. Retrieved Jun. 5, 2011, from http://www.soapui.org

Sotomayor B., Montero R. S., Llorente I. M. & Foster I. (2009). An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds, *Internet Computing*, 13(5), 14-22.

Sukhoroslov, O. V. (2009). JLite: a lightweight Java API for gLite. Retrieved Jan. 25, 2012, from http://jlite.googlecode.com/files/jLite.pdf

Taylor I., Shields M., Wang I. & Harrison A. (2005). Visual Grid workflow in Triana, *Journal of Grid Computing*, 3 (3-4), 153-169.

TORQUE (2012). Adaptive Computing Torque Resource Manager. TORQUE Administration Guide v2.5.9. Retrieved Jun. 5, 2012, from http://www.adaptivecomputing.com/download/resources/docs/torque/2-5-9/pdf/TORQUE_Administrator%27s_Guide.pdf

Tröger P. (2011). DRMAAv2 - An Introduction. Retrieved Jan. 23, 2012, from http://www.drmaa.org/drmaav2-ogf33.pdf

Vàzquez C., Huedo E., Montero R. S. & Llorente I. M. (2011). On the use of clouds for grid resource provisioning, *Future Generation Computer Systems*, 27, 600-605.

W3C (2001). Web Services Description Language 1.1. Retrieved Jun. 9, 2011, from http://www.w3.org/TR/wsd