DOI: http://dx.doi.org/10.1016/j.future.2017.02.047

Strategies and Systems towards Grids and Clouds Integration: A DBMS-Based Solution

Mirko Mariotti[a,*], Osvaldo Gervasi[b], Flavio Vella[c], Alfredo Cuzzocrea[d], Alessandro Costantini[e]

[a]*Department of Physics and Geology, University of Perugia, Via Pascoli - 06123 Perugia, Italy*
[b]*Department of Mathematics and Computer Science, University of Perugia, Via Vanvitelli, 1 - 06123 Perugia, Italy*
[c]*Department of Computer Science, University of Rome La Sapienza, Viale Regina Elena, 295 - 00198 Rome, Italy*
[d]*DIA Department, University of Trieste and ICAR-CNR Piazzale Europa 1 - 34127 Trieste, Italy*
[e]*CNAF-INFN, Viale Berti Pichat 6 - 40127 Bologna, Italy*

Abstract

Cloud and Grid computing share some essential driving ideas although the computing and economic models are very different. In this paper, we propose different strategies for the Batch-oriented and Service-oriented computing models interoperability. In particular, we describe an innovative approach to connect together Computational Grids and IaaS providers. This is achieved via introducing a simple and powerful DBMS-based system of deploying VM images from a Cloud environment in order to fulfill particular requests of task execution coming from a Grid environment. From a user point of view, resource authorization and access are kept unchanged, thus preserving the user experience related to the Grid. From the accounting point of view, in order to inform the Grid sites that a certain resource is available on a given Cloud-enabled Grid site, the information is published on the Grid information system. In this so-delineated scenario, we are able of using the powerful capability of distributing jobs of the Grid in order to allocate resources not only belonging to Grid clusters, but also with different architectures like GPUs, FPGAs and other systems. The target DBMS-based system has been designed for orchestrate a set of computing systems able to provide physical and virtual resources, creating a unified system, in which the various users-submitted computing tasks are managed and optimized. The goodness of the proposed system is demonstrated by a series of experiments

*Corresponding author
Email addresses: mirko.mariotti@unipg.it (Mirko Mariotti), osvaldo.gervasi@unipg.it (Osvaldo Gervasi), vella@di.uniroma1.it (Flavio Vella), alfredo.cuzzocrea@dia.units.it (Alfredo Cuzzocrea), alessandro.costantini@cnaf.infn.it (Alessandro Costantini)

highlighting the benefits of our approach.

Keywords: Resource integration, Cloud computing, Grid computing, Distributed Environments, Heterogeneous environments, Multi/many core computing, GPGPU computing

---

1. Introduction

Nowadays two trends are evident in the way the computational resources are organized and managed to provide users the computing infrastructures adequate for the emerging computing needs. On the one hand, virtualization technologies are massively adopted, based on more and more powerful Cloud systems (Openstack, Opennebula, Eucaliptus, etc.), along with systems for deploying Virtual Machines and all technologies related to the Cloud scenario. On the other hand, it is clear that in order to increase the performances of the computing systems the best way is to adopt heterogeneous architectures, specializing them on the basis of the requested type of computation from the users. Examples of this type are the usage of GPUs for the fast solution of different problems in computer science like graph analysis [10, 11], cryptography [22, 39, 19] or computational logic[18], the development of innovative architectures, like the Parallella board[32], and the adoption of Field-Programmable Gate Array (FPGA) device as a computing resource[27, 13]. Cloud and Grid computing share some essential driving ideas which led to the construction of both large scale federated Grid infrastructures which can be summarized as follows:

- bring the promise of encapsulating the complexity of hardware resources and make them easily accessible by means of high-level user interfaces;
- address some form of the intrinsic scalability issues of large scale computational challenges;
- cope with the need of resources that cannot be hosted on premises.

However, the key differences between Grids and Clouds concern abstractions and compute models adopted by both paradigms [21]. It can be said that Grids are built "bottom up" and are concerned more with federation of static existing resources that typically are legacy clusters built around a Local Resources Management System (LRMS) that exploits the Batch computing model.

The development of applications for Grid environments requires the knowledge of the Grid infrastructure abstractions. This process, aimed at enabling the application to run in such environments, is in fact called "Grid-enabling" and can be rather complex [20], even considering real-life systems (e.g., [16, 25]). On the other hand, Cloud users can choose their own compute model, leveraging more general (without the needs of a fine tuning of the environment) interfaces that often lead to simpler interaction and application development [36].

As sketched before, there are several problems that do not play nicely with the heterogeneous nature of aggregated resources in Grids. For example, many

scientific applications need different environments (operating systems, libraries) and hardware (i.e., Multicore Processors, FPGA, GPUs, etc.). From a Batch point of view this represents a set of requirements influencing scheduling decisions for both top and local level resource managers. So the Grid sites heterogeneity plays a central role in job distribution (workload) among sites that match the aforementioned requirements.

Furthermore the Grid workload can be often unpredictable and subject to burst increase, that lead to unbalanced distribution in resource usage, and even deterioration of QoS. In this context the Grid workflow represents a weak point for the Batch model in which resources are often statically managed and partitioned, and cannot be adapted in advance to meet possible requirements. Moreover the use of Clouds could allow the extension of private resource pools in number and typology with positive effects on Quality of Service (QoS).

So why do not dismiss Grids and adopt Cloud solutions? There are several reasons: it is not yet clear how some critical issues (data management, security, etc.) are to be dealt with in the Cloud era, while in Grid are well-established. Furthermore, the costs of an eventual shift in technology must be thoroughly investigated. A more reasonable approach is an integration process that combines the features of both. The latter one is the main focus of our work. To this end, we introduce a simple and powerful DBMS-based system of deploying VM images from a Cloud environment in order to fulfill particular requests of task execution coming from a Grid environment. From a user point of view, resource authorization and access are kept unchanged, thus preserving the user experience related to the Grid. From the accounting point of view, in order to inform the Grid sites that a certain resource is available on a given Cloud-enabled Grid site, the information is published on the Grid information system. In this so-delineated scenario, we are able of using the powerful capability of distributing jobs of the Grid in order to allocate resources not only belonging to Grid clusters, but also with different architectures like GPUs, FPGAs and other systems. The target DBMS-based system has been designed for orchestrate a set of computing systems able to provide physical and virtual resources, creating a unified system, in which the various users-submitted computing tasks are managed and optimized. The goodness of the proposed system is demonstrated by a series of experiments highlighting the benefits of our approach.

This paper is organized as follows: in Section 2 the related work and different integration possibilities are illustrated; after a brief overview of the proposed solution (Section 3), in Section 4 we describe the internal organization of the main DBMS component where the system is built around. In Section 5 and Section 6 the software components and the job flow are presented and discussed. In Section 7 the *Direct2M* component is described. Finally, in Section 8 the results are discussed and in Section 9 some conclusions and the future work are addressed.

2. Related Work and Integration Possibilities

Before the Cloud era, even if these issues were addressed in various works[12, 38, 28], the proposed solutions were often heavy customized and too tightly dependent on particular technological choices.

With the success of Cloud computing through the spread of IaaS providers [8, 3, 4], the development of interfaces for the simplification of virtual management [5, 7] and related libraries (i.e.,[6, 2], etc.) paved the way to several possibilities for Grid and Cloud integration. As a matter of fact, even if Cloud solutions have been, since their first definition[14], primarily driven by business motivations, the IaaS service model seems to respond to some of the Batch model issues and can overcome them with both on-demand and adaptive characteristics.

To the best of our knowledge there are three approaches of site-level integration between the Batch oriented Grid compute model and the service oriented nature of Clouds.

Grid over Clouds. In this model, a whole Grid site is built on top of a public/private Cloud. Through this schema, the Grid infrastructure can be built by instantiating resources according to the real needs of the users. In [9], the authors provided a "Grid as a service" tool in order to create new Grid sites, or to add computational resources to existing Grid sites by exploiting a Platform as a service (PaaS) approach. Similar approach is also adopted in [30].

Hybrid with Batch-dependent Cloud-enabled LRMS. In this model, a single local Batch system is used to schedule the jobs on a pool of dynamically provisioned resources either on premises or public/private Clouds. For example, in [35], the authors described a solution which enables building dynamical environments through Grid jobs or local Cloud jobs. The solution proposed is built around the LRMS which handles each request. This approach presents manifold limitations. The main drawbacks are the following: *i*) the solution is strongly dependent on a particular technology adopted (i.e. LRMS requires customizations); *ii*) the approach is not *elastic* [33, 31] since it enables the spawn of a virtual environment on local resources only.

Hybrid no Batch-dependent. In this model, the local Grid site spawns resources (even whole clusters) on public/private Clouds on the basis of the jobs requests. The integration is done at the Computing Element level. In this way, several computational resources (i.e. resources available on other computational centers) can be exploited by a fine grained control over virtual instances. In [39], the first solution based on Cloud-over-Grid approach was presented. The authors also validated their solution providing to Grid users special virtual computational resources as GPUs.

The last two approaches can be also identified as two types of "Cloud-over-Grid". In the present work, we describe our solution, that may be used to implement any of the described hybrid approaches with the special attention to the no Batch-dependent model.

3. A DBMS-Based System for Integrating Grids and Clouds: Overview

The proposed system is based on the adoption of Cloud systems to enable the Grid sites to provide to the users a set of non-traditional resources, like the aforementioned ones and dynamical environments. As an example a Grid user may request to run in a server equipped with a given GPU, or with a particular software library installed or operating system.

Our system has been designed according to the Unix principles: each component is autonomous, independent from the others, specialized in carrying out a named task in a simple way. According to this approach we have chosen to use tags for cataloging the Virtual Machines that have a certain type of hardware features (such as a named architecture, hybrid systems, GPU, etc.) or software (operating systems, special libraries installed). These tags are published using the standard techniques of the Grid environment, as features implemented and published by a particular site, and which can be specified as requirements by the users when submitting a job. In this way, the Grid information system enables the users to submit jobs requiring special environments, provided only by some Grid sites.

In Figure 1 the project logical schema is sketched. Our solution is built around a DBMS that plays a central role since it contains the configuration of the system, in terms of the connected clusters, the Cloud systems, and their environments and status. The architectural workflow of our solution is implemented by different agents connected to the DBMS each performing a specific action; they will be described in detail later.

In the remaining part of this paper we will use the following terms, and corresponding meaning:

Computing Element (CE): is the set of resources made by the *Gatekeeper* and the *Cluster*.

Gatekeeper: is the system that provides the gateway through which the Grid jobs are submitted to the Batch system running on the local farm nodes;

Cluster: it is a Grid enabled Cluster, i.e. a bunch of *Worker Nodes* (WNs), connected to a Computing Element (CE) and connected to the Grid system. When referring to Clusters we will mean the Cluster Resource Manager.

Cloud: it is a Cloud infrastructure with a Cloud controller like OpenNebula, OpenStack or Eucalyptus.

Computational node: a single server used as target of the incoming Grid job without the use of a Batch system or a Cloud System.

Cloudtag: tag used to mark the Virtual Machine (VM) images and to organize the infrastructure resources.

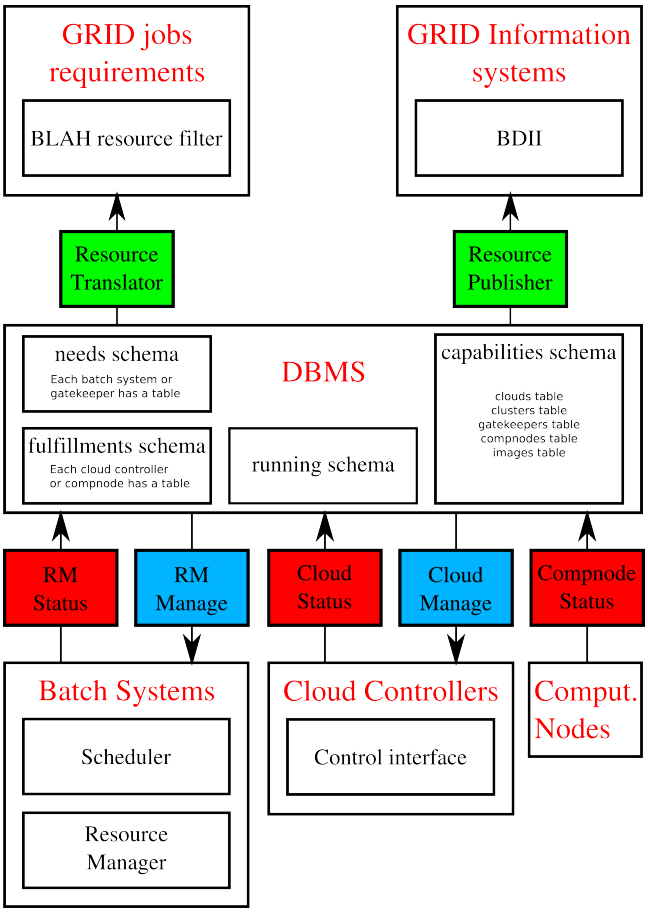


Figure 1: Description of the proposed system

4. The DBMS Structure

The information about Clusters and Clouds is collected on a DBMS system. From the implementation point of view we have chosen PostgreSQL for this purpose. The information has been divided into four logic blocks, each one mapped to a DBMS schema:

- The *capabilities* schema contains the information about the Clusters, Cloud Controllers and VM images known to the system. It also contains the information about tagging the VM images to publish this information through to Grid information system.
- The *needs* schema contains a live view of the *Cloudtag* needed by clusters.
- The *fulfillments* schema contains a live view of the *Cloudtag* offered by Cloud systems.

- The *running* schema contains the list of the running jobs, with related details.

4.1. The CAPABILITIES *Schema*

The *capabilities* schema contains the information related to the composition of the various systems. Each software component reads from the DB the necessary information, since the *capabilities* schema contains all the information related to the structure of the system. The information contained in this schema concern the Cloud, Clusters, Gatekeepers, the Computational nodes connected to the system, and, more important, the VM images.

4.1.1. Information on the Active Cloud Systems

The table *Clouds* of the *capabilities* schema traces the following information related to the active Cloud systems:

- Type of Cloud system (i.e., OpenStack, OpenNebula or Eucaliptus).
- The description of the Cloud system.
- The information on how to interact with the system, which may, or may not, contain authentication information.

4.1.2. Information on the Active Clusters

The table *clusters* of the capabilities schema traces the following information related to the Batch system of the active Clusters:

- Type of Cluster's Resource Manager (Torque/MAUI, LFS, etc).
- The description of the Cluster.
- The optional information on how to interact with the Batch system of the Cluster.

4.1.3. Information on Gatekeepers

The table *gatekeepers* of the *capabilities* schema traces the information related to the Gatekeepers of the active Grid nodes. In particular, the more important information are:

- Gatekeeper information and the Information System of the Grid site.
- Description of the Grid site.
- The optional information on how to access the Gatekeeper and/or the Information System.

It is relevant to notice the reason why we implemented two separate tables, one for Gatekeepers and one for the Batch Systems, even if the Grid site is the same, so that the CE is listed in the *gatekeepers* table and the Batch System in *clusters* table. We kept separated the two tables because we want to stress the fact that the job path, and the related sequence of events and actions, are different if they are under the control of a Batch System or not.

4.1.4. Information on Standalone Computational Nodes

The table *compnodes* of the *capabilities* schema traces the information related to the access to Computational nodes capable of executing jobs. They represent the real or Virtual Machines not connected to a Batch System, we want to include in our System. The most relevant information are:

- Node type.
- Operating system.
- Information related to the access to the node.

4.1.5. VM Images

A job can be received by a Gatekeeper or by a Batch System and sent to a virtual resource (Cloud) or on a standalone Server or Computational node. The aforementioned resources can be of two types: those that require the fulfillment of a need (Gatekeepers and Clusters) and those that satisfy the need (Clouds and Server). The association between requests to meet and who can satisfy them is performed inside the table *images*.

The possible job flows originated by this schema are four and are described in Table 1. The single flows will be discussed in the next sections.

Table 1: Possible job flows

Component	**Target**
Cluster	Server
Cluster	Gloud
Gatekeeper	Cloud
Gatekeeper	Server

The table *images* contains the couple of values *Component* from which the job is coming and *Target*, indicating the job flow in the system, the tag that will be published by the Grid Information System in order to notify the presence of the resource, and a series of information related to the possibility of creating multiple instances. In particular are advised the following information:

- How many instances may be generated (for a single Computational node this value is 1).
- Number of jobs per instance.
- Magnitude and boundaries of the instances.
- Waiting time before destroying the images.

4.2. The NEEDS *Schema*

In the *needs* schema the software agents running on Clusters and/or Gatekeepers connected to the system, maintain the status of requests to be satisfied. Each agent has associated a table containing the list of jobs with the related *Cloudtags*. The job listed in such tables are all waiting jobs. Running jobs are listed in the *running* schema.

4.3. The FULFILLMENTS *Schema*

In the *fulfillments* schema are instead listed the resources available to satisfy the requests, so that the systems may know for each *Cloudtag* where is located the Cloud or the Computational node.

4.4. The RUNNING *Schema*

We included in the system also the *running* schema having the purpose of storing the state of running resources.

5. Software Components

The implemented system is based on the DBMS and on a set of daemons, each of them carries out a limited and specific action. We can identify three functional classes:

- Daemons that integrate the information on a Computational node, a Cluster or a Cloud or on the Grid. They create a global view of the system, that is crucial to let the single components to interact properly each other. They also publish the information related to a component on another one: i.e., a given *Cloudtag* offered by a Cloud has to be propagated via the Grid, in order to be accessed from remote users in a secure way.

- Daemons that keep updated on the DBMS the state of a system (Cloud, Cluster or Computational nodes). It represents the passive part of the system, collects the utilization information related to the Cloud and Grid systems and insert them into the database.

- Daemons that perform actions on the Cluster and on the Cloud, when necessary. They represent the active and decision-making part of the system. Based on the current status information may for example decide to instantiate new virtual resources to meet the needs of a single cluster. As an example, a given component may suspend a job because it is not yet available the resource where it has to run.

5.1. Resource Publisher

The Resource Publisher components are responsible for the publication of the resource capabilities within the site Berkeley Database Information Index (BDII) of every cluster connected to the system. These agents, typically one per Grid site connected to the system, read from the database the set of *Cloudtag* and modify the site information system regarding the available resources, to advice their presence. In our case, we implemented the interface agent with the Grid agents based on Glite middleware. The agent inserts each *Cloudtag* supported into the BDII of the Grid site, in the LDAP directory.

5.2. Resource Translator

The presence of *Cloudtag* as supported features of a Grid site, enables the users to specify in the requirements for a job, one of such tags. When the job arrives in the Grid site it is necessary to recover the information on tags and utilize them for letting the job continue the execution in the right way. An agent takes charge of modifying the Grid site information related to the supported *Cloudtags*. In our case we developed a Grid agent based on Glite middleware.

5.3. Grid Connectors

The purpose of Grid Connectors is to maintain, for each Cluster, the status of the *Cloudtag* requests in the database table related to such Cluster. The system needs an active connector for each Cluster connected to the system and such connectors are dependent from the type of Resource Manager installed on the Grid site. We implemented the Torque/MAUI connectors. Similarly it is possible to implement the connectors for the other systems (LSF, Condor, etc).

5.3.1. Torque/MAUI Connector and Advisor

Upon job arrival another agent instructs the scheduler MAUI not to start the job, at least at the beginning. This is done to avoid the job failure in case one of the necessary resources to run the job is not yet ready. The job is then blocked on the Batch system maintaining the *Cloudtag* attribute.

The connector for Torque Batch system then tracks jobs within the resource manager that has some *Cloudtag* listed as required resource, and updates the *needs* table belonging to the cluster where it runs. This is done by a C program directly interfaced with the Torque server.

Another agent, checking the running status and the *needs* schema, takes the decision about deploying or destroying virtual resources and unlocks, if necessary, the frozen jobs.

With this model the Resource Manager still is in charge of executing, scheduling, prioritizing jobs, however the resource pool is modified by our agents according to the needs expressed by the *Cloudtags*.

5.4. Grid Advisors

Sometimes, it happens that the behavior of a managed Grid site has to be changed. If a job needs for example a resource not yet ready, we may force the Batch System to hold the job. If a job has to run on a Computational node instead of being managed by the Batch System, we have to launch the proper scripts, acting differently respect to the standard behavior. The aforementioned examples show as our system modifies through Agents the standard behavior of a Grid site to obtain the desired result.

5.5. Cloud Connectors and Standalone Computational Node Connectors

As is the case of the Grid sites, also for the Clouds connected to the system, a group of agents (usually one per Cloud System) maintain a copy of the virtual resources status of the named Cloud, related to the system. The same occurs for the standalone Computational nodes connected to the system.

In order to use the most common Cloud Systems, we implemented the following Agents, using the following languages and libraries:

- OpenStack: the Fog library and the Ruby language;
- OpenNebula: the Boto library and the Python language;
- EC2 generic: the Boto library and the Python language.

5.6. Cloud Advisors

Another relevant class of Agents in our system is that taking charge of instantiate, delete, or modify virtual resources on the managed Clouds. Reading from the DBMS the current status of the resources and respecting a set of rules defined in their configuration, these daemons decide when create or destroy virtual resources. In this case also, we used the languages listed in Section 5.5.

6. Job Flow

Considering what has been said so far, in our system are implemented three different job flows:

- Jobs arriving from the Batch System that are executed on a virtual resource provided by a Cloud System;
- Jobs arriving to the Gatekeeper that are not transferred to the Batch System and instead are moved on a virtual resource;
- Jobs arriving to the Gatekeeper that are not transferred to the Batch System and instead are moved on a real resource, not managed by a Cloud System.

The flow related to a job transferred to a Batch System and then to a real machine, represents the standard behavior of a Grid site and therefore will be not considered in this paper.

6.1. Cluster and Cloud

The first considered job flow implemented in our system is related to a job arriving via Grid to the Gatekeeper and is transferred (with its *Cloudtag*) to the Batch System. Up to this point our system translates the tags expressed in the Grid format to the corresponding Batch System tags. An Agent verifies then, crossing the live data of jobs in the waiting queue and related *Cloudtags*, with the data of the Cloud System, if it will be necessary to instantiate resources on the Cloud System. Similarly, the Agent may decide to remove a Virtual Machine from the Cloud System. In Figure 2 it is sketched such type of job flow.

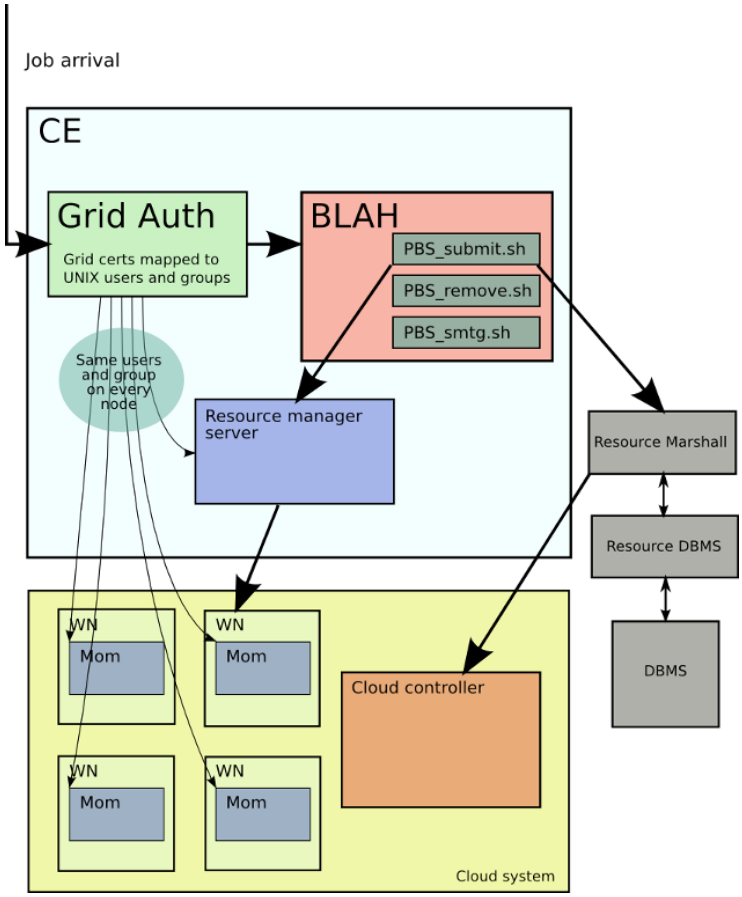


Figure 2: Flow related to Batch System and Cloud.

6.2. Gatekeeper and Cloud

The second job flow implemented in our system is related to a job arriving to the Gatekeeper and not transferred to the Batch System, but enters the

Direct2M (D2M) component, that has been implemented by us. The *Direct2M* component temporarily blocks the job and verifies if exists a virtual resource available, matching the *Cloudtag*. If the resource is available, *Direct2M* acquire the necessary authentications and authorizations and executes the job in the Virtual Machine. If the resource is busy the job is blocked until the resource if released or another resource which supports the *Cloudtag* may be instanced.

Figure 3 shows the diagram related to such a flow.

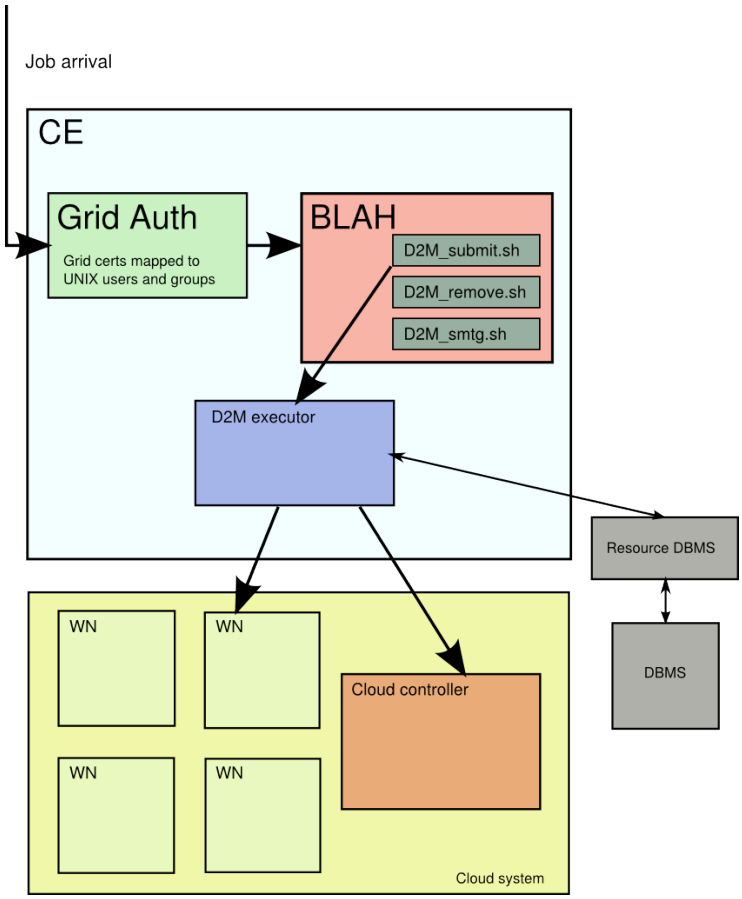


Figure 3: Flow related to a Cloud resource and no Batch System.

6.3. Gatekeeper and Server

The third flow we considered is related to the following scenario: the incoming job is blocked on the Gatekeeper and is transferred to a Computational node. In such cases the Cloud component is not active. Since the system cannot instantiate new resources, the Agent function is that of unblocking the job and

send it to a resource as soon as the resource becomes available. The Figure 4 shows such a scenario.

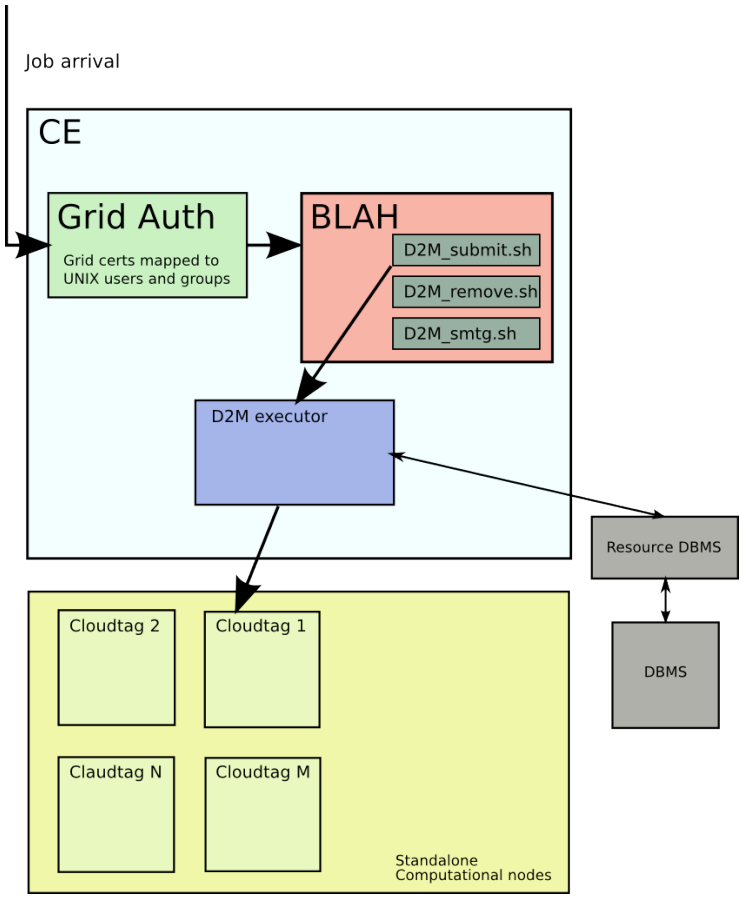


Figure 4: Flow related to a Cloud free resource without Batch System.

7. Direct2M Component

The implemented system is based on the hypothesis that Virtual Machines used as Worker Nodes are included in the pool of resources the Gatekeeper of the Grid site manages. This means that all typical operations carried out by the Grid middleware (move the job binaries to the right place, execute the job with the proper ownership, the verification of authorizations, the collection of results, etc.) are delegated to the Resource Manager of the Grid System. This imply that the Grid middleware has to be installed into such Virtual Machines managed by the Cloud System. However in some circumstances, the architectures and /or the Operating Systems are to much different from the ones supporting

the middleware. In example, we cannot run a job in a Virtual Machine running an OpenBSD System, since it does not support Glite middleware. To make it possible, we developed an agent to enable the job execution on system without the Grid middleware installed, named *Direct2M*.

A set of heterogeneous resources (either virtual or real), not included in the Grid Resource Manager, enabled with *Direct2M* component, can execute Grid jobs and some functionalities usually carried out by the Grid Resource Manager are performed by this component. Extended Gatekeeper features are also necessary in order to execute the *Direct2M* component. In our case studies, we modified the Glite middleware, in particular we added the following capabilities to the Gatekeeper:

Job management at Gatekeeper level: since when the *Direct2M* component is used we are implementing a "no Batch-dependent model" we create some bash scripts to get and control the jobs. This component is interfaced with the Batch Local ASCII Helper (BLAH) component of the Glite middleware.

User and group mapping: every job authorized by the Grid is mapped on the Gatekeeper to a string representing the Person/Organization of the job owner. This string is then mapped to a Gatekeeper local user. We needed to map this user to a local one of the target machine.

Data moving from and to servers: we choose to use `ssh` to move input data of jobs, retrieve outputs and as a method to submit the job. A set of control scripts have been written to carry out all these operations.

8. Experimental Results

By using the Grid infrastructure of the Italian Grid Infrastructure available in the Department of Mathematics and Computer Science of the Perugia University, we implemented all software components necessary to integrate the Grid site with the most popular Cloud Systems: OpenStack and OpenNebula. We built an experimental Grid site connected to an OpenNebula and an OpenStack Cloud systems. As for the Computational nodes, we included some Linux systems and some Unix systems, Illumos based distribution and some BSD flavors. We also tested the system with no-x86 systems: notably raspberryPI, Beaglebone Black systems.

A user who access the implemented system has the opportunity to exploit, using a standard Grid interface, heterogeneous resources, both in hardware and software components.

Concerning the performance point of view, our system introduces a negligible overhead when it is used as "Hybrid with Batch-dependent Cloud-enabled LRMS" and the resources are deployed in advance. If the resources have to be deployed, the overhead introduced by our system is negligible, compared to the time necessary to deploy the resources. When used as "Hybrid no Batch-dependent model" an additional overhead is introduced by the `ssh` command, used to move data and launch jobs.

8.1. Case Study

To better understand the implemented system, a complete example will be now illustrated. Let's assume of having a particular library, named `cgintlib`, installed in an image of a Cloud system connected to the computing environment described so far. The goal is to enable jobs needing `cgintlib`, submitted to a Grid owning the site connected to the implemented environment, to be executed on computing resources provided by the aforementioned Cloud.

In this example we will consider the OpenStack Cloud Infrastructure, the Grid environment EGI, and as per the VM image, it will not be part of the resources controlled be the Grid resource manager, so that the working scenario is the "Hybrid no Batch dependent" (see Section **??**). Let's assume also to have a working image with the `cgintlib` library installed, and that the OpenStack installation is configured to spawn an instance of this image upon request. Three records in the DMBS instruct the system to handle this scenario:

1. A record in the *capabilities schema* table named *Cloud* tells the system the available Cloud controller that can provide the required resources and the necessary information to access it. In the present example the record will contain the Cloud controller type (OpenStack), its IP address, the authentication information, etc. There will be a record for each available Cloud controller.
2. A record in the *capabilities schema* table named *gatekeepers* tells the system that exists a Grid enabled gatekepeer which may accept jobs to be executed in the implemented environment. In the present example the record will contain the Grid environment type (EGI), its Gatekeeper IP address, etc.
3. A record in the *capabilities schema* table named *images* instructs the system that exists an image that implements the `cgintlib` Cloudtag (we chose the same name for the library and for the Cloudtag), and selects the proper job flow for this image. In the present example the flow is Gatekeeper $\rightarrow$ Cloud meaning that the jobs will not be forwarded to the Grid site Batch system but will be handled by the *Direct2VM* component. The flow is inferred by the two references to the two previous records contained in the present record.

This information, and the proper VM image configuration, are the only things needed to make the system working. Once the system is configured with the previous information, firstly the Grid site publishes the information about the `cgintlib` capability and is reconfigured to handle requests; secondly upon job arrival the request is processed. Let's examine these two processes in detail in the present example.

When the resource publisher running in the Grid site argues the existence of a new capability (in the form of the Cloudtag `cgintlib`) within the capabilities schema, it publishes this capability in the Grid DBII as supported Software Runtime Environment. The Grid Workload Management System will then be able to serve a requirement specified in the user's JDL file, allocating the Job

in the Grid site which published the proper capability. The resource publisher demon publishes the information according the capabilities schema.

Since the job will not make use of the Grid local resource manager, two other operations are requested in order to handle jobs arrival. The BLAH parser is aware that jobs requesting `cgintlib` do not have to be send to the Batch system and that an alternative environment setup is requested. All these actions are performed by the Grid site resource translator, which firstly deploys the Direct2VM BLAH script (opposing to the one already present to handle various resource managers), and then modify the BLAH parser to target these scripts and finally runs a D2M executor, a demon whose purpose in to talk with the IaaS provider and its VM instances.

The described scenario enables the system to handle the jobs tagged as `cgintlib`. Upon job arrival the BLAH parser matches the Cloudtag `cgintlib` and, since the target environment is the Cloud environment, it transfers the control to the Direct2VM BLAH scripts. The job is now blocked on the Gatekeeper, the D2M executor is informed about the job, and the proper table within the *needs* schema is updated. The D2M executor firstly matches the entries in the capabilities schema, to identify the target environment (in the present case a Virtual Machine in a OpenStack Cloud, however, several types of Cloud environments and even standalone servers are supported). It matches the tables of the *fullfilments* schema with those of the *needs* schema to verify that the given target already has a running VM suitable for the job. Depending on the cases it will either spawn a new VM instance, remove a running VM instance, or wait until some resource will be freed. The *fulfillments* schema has a table relative to the given OpenStack Cloud system and a Cloud Status daemon will keep the information updated.

When the necessary resources are available for job execution, the job is unblocked, executed and finally its results are moved back to the Gatekeeper. During these phases the D2M executor also updates the tables of the *running* schema. All these operations are the result of the interaction between the D2M executor and the Direct2VM BLAH scripts.

8.2. Experiment Assessment Details

We run four experiments to prove the goodness of our method. To this purpose, four groups of 100 jobs have been submitted to the system to measure its response time.

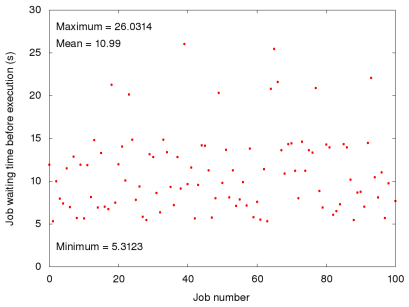
To avoid to measure a systematic delay due to the GRID middleware, we considered the job active when it reached the gatekeeper and not when it was submitted. Moreover, the job is considered done, when its execution is starts. The other assumption made during these tests was that the resources are always idle and no additional delay should be introduced because the jobs are waiting. Our tests are presented in the next sections.

It should be noted as our proposed test-bed focusing on 100 jobs could also be extended to a larger number of concurrent jobs. Indeed, since our main goal consists in providing a milestone-study on showing how the proposed DBMS-based system for supporting Grid and Cloud integration works, we firmly retain

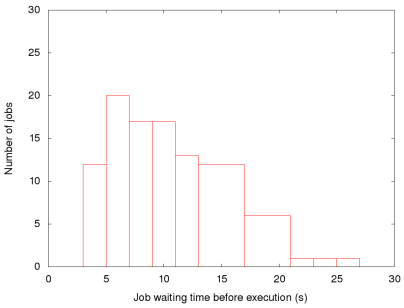
that this setting already provides a sufficient experimental environment where highlighting pros and cons of the proposed solution and, above all, precisely detecting how the actual integration layer (of our system) impacts on both the target Grid and Cloud systems, respectively. Our final experimental results, as clearly demonstrated in the next Sections, completely fulfill this goal. On the other hand, extending experimental work as to further stress the scalability of the system is left as future work.

8.2.1. Cluster to Server Example

All jobs are dispatched to the Local Resource Manager (Torque/Maui); this type of flow represents the standard flow of a Grid environment. In Figure 5(a) the Job waiting time is shown (Minimum time: 5s, Maximum time: 26s, Mean waiting time: 11s), while in Figure 5(b) is shown the jobs distribution among the waiting time (boxed by 2 seconds intervals). Such experiment measures the latency of the Grid middleware, since it is related to the dispatch of a Job in a Grid site on its Local Resource Manager.



(a) Waiting times before execution of the set of jobs.

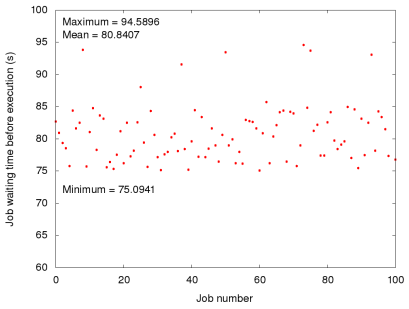


(b) Job distribution (boxed by 2 seconds intervals) respect to the waiting time before execution.

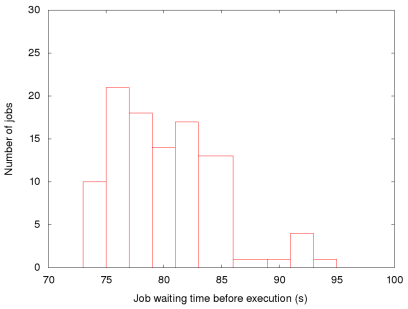
Figure 5: Results obtained measuring the waiting time before execution of a set of 100 jobs in the *Cluster to Server* Job flow.

8.2.2. Cluster to Cloud Example

All jobs are dispatched to the Local Resource Manager (Torque/Maui) and the Worker Node is started on the Cloud system; this type of flow was previously described as *cluster to cloud* in Section 6.1. In Figure 6(a) the Job waiting time is shown (Minimum time: 75s, Maximum time: 95s, Mean waiting time: 81s), while in Figure 6(b) is shown the jobs distribution among the waiting time (boxed by 2 seconds intervals). Such experiment measures the latency of the Grid of the previous experiment, plus the time necessary to instantiate a Virtual Machine on the Cloud system (70s, approximately). In this experiment, we ignored the time necessary to close the instance of the Virtual Machine.



(a) Waiting times before execution of the set of jobs.

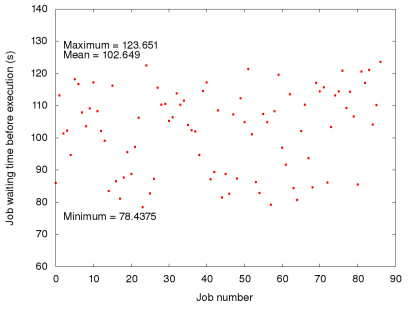


(b) Job distribution (boxed by 2 seconds intervals) respect to the waiting time before execution.

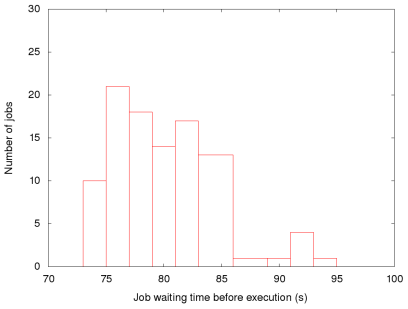
Figure 6: Results obtained measuring the waiting time before execution of a set of 100 jobs in the *Cluster to Cloud* Job flow.

8.2.3. Gatekeeper to Cloud Example

All jobs are handled by the *Direct2M* components targeting a Virtual Machine image on the Cloud system, this type of flow was previously described as *gatekeeper to cloud* in Section 6.2. In Figure 7(a) the Job waiting time is shown (Minimum time: 78s, Maximum time: 124s, Mean waiting time: 103s), while in Figure 7(b) is shown the jobs distribution among the waiting time (boxed by 2 seconds intervals). Such experiment measures, if compared with the previous one, the effect of the interaction with the *Direct2M* component (22s, approximately), before the Virtual Machine will be instantiated.



(a) Waiting times before execution of the set of jobs.

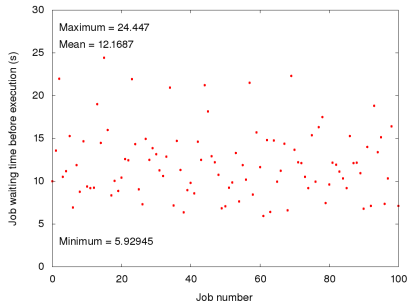


(b) Job distribution (boxed by 2 seconds intervals) respect to the waiting time before execution.

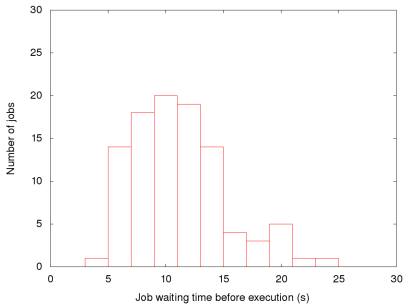
Figure 7: Results obtained measuring the waiting t3me before execution of a set of 100 jobs in the *Gatekeeper to Cloud* Job flow.

8.2.4. Gatekeeper to Server Example

All jobs are handled by the *Direct2M* components targeting an image on a real server; this flow is described as *gatekeeper to server* in Section 6.3. In Figure 8(a) the Job waiting time is shown (Minimum time: 6s, Maximum time: 24s, Mean waiting time: 12s), while in Figure 8(b) is shown the jobs distribution among the waiting time (boxed by 2 seconds intervals). In this experiment the data distribution is similar to that of the *cluster to server* case, with some overhead due to the facts that i) the job is always stopped and restarted by the *Direct2M* component and ii) its processing is made using SSH that is slower than a proper batch system (i.e., Torque).



(a) Waiting times before execution of the set of jobs.



(b) Job distribution (boxed by 2 seconds intervals) respect to the waiting time before execution.

Figure 8: Results obtained measuring the waiting time before execution of a set of 100 jobs in the *Gatekeeper to Server* Job flow.

8.3. Use Case: Enabling GPU Computing

Nowadays, GPUs are widely used high-performance devices to accelerate scientific applications. Moreover, virtualization technologies allows obtaining reasonable performance with respect to real machines (see for example [40]), supporting a wide range of different architectures. For example, Xen hypervisor is fully compliant with x86, x86_64, IA64 and ARM architectures. It also provided the support for the virtualization of input/output memory management unit (IOMMU) of the motherboard chip-set that enable the use of direct memory access (DMA) to I/O bus is also necessary to efficiently use peripheral devices in guest virtual machines. When those hardware prerequisites are met Xen permits to transparently assign PCIe devices like VGA devices to VMs. This assignment can be performed at the creation of the Virtual Machine or at runtime but a single PCIe resource can not be shared between VMs ensuring is this way the exclusive access of hardware resources.

Concerning GPUs virtualization, graphic adapters support many legacy x86 features that must be provided in the virtual environment. Qemu-dm emulator used by the Xen HVM guests, needs to disable the internal (emulated) graphics

adapter, and copy and map the real graphics adapter VGA BIOS to the Virtual Machine memory.

A different approach is followed when the graphic driver is splitted into a front-end component, deployed on the Virtual Machine, and a back-end one, deployed on the real machine, setting a communication channel between them [23]. This technique also allows sharing of GPUs between multiple Virtual Machines.

Other notable virtual GPU solutions are provided in [26, 24] and [37] as well. We tested our system on a motherboard which support both Intel Virtualization Technology (VT-x) and Intel Virtualization Technology for Directed I/O (VT-d). The server is powered by an Intel i7 870 CPU and Nvidia GeForce 580 (Fermi). In our test, we compared the performance of real GPU and virtual GPU provided by means of PCIe pass-through on Debian operating system with XEN hypervisor.

The samples provided by Nvidia developer kit are used as benchmarks. We performed two different experiments. In the first one we measured the bandwidth during memory transfer over PCIe. Furthermore, we evaluated the performance of both real and virtual GPU by performing arithmetic-intensive operations like matrix multiplication.

Concerning the bandwidth tests, we performed Host-Device memory transfers. The memory was pinned and the size was varying from 1 Byte to 64 MB. Data transfer over PCIe should represent the most relevant source of virtualization overhead, since it requires interrupt remapping, IOMMU interaction, etc., in order to enable PCI pass-through into the VM. Despite this, we observed a maximum of 2% overhead.

Finally, concerning arithmetic intensive evaluation we performed a squared matrix multiplication. As shown in Figure 9 the Virtual GPU (in red) is about 4% slower than Real GPU (in blue). Similar results are also reported in [40] by using GPU pass-through and Xen hypervisor.

As per the Grid side, a user can require a GPU specifying such resource via the Job Description Language which is usually defined for a specific Grid middleware. In Listing 1, an example is provided for the EGI environment [1].

Listing 1: *JDL example with GPU flavor*

```
Type = "job";
executable="";
[...]
CeRequirements="other.GlueHostMainMemoryRAMSize > 2048 &&
(Member(\"GPU\",
other.GlueHostApplicationSoftwareRuntimeEnvironment))";
```

9. Conclusions and Future Work

In the this work, we presented different integration strategies which allow a simple interoperability between Batch-oriented and Service-oriented computing models.

Computational Grids provide simple user interface for the execution of scientific applications (Batch jobs) through hierarchical components exploiting statical pre-allocated computational resources. On the other hand, Cloud computing allows exploiting elastic computation by means of virtualization technologies. Through this approach the user is able to use heterogeneous and specialized computation resources like GPUs, FPGA or other embedded systems transparently.

We provided a straightforward implementation of one of the proposed strategies, which finally determines a novel DBMS-based system for the integration of Grids and Clouds. The implemented system allows the utilization of the Grid and its standard utilities for publishing the information related to a given Grid site, for accepting and executing jobs requiring particular software tools, libraries and specific hardware characteristics. The system, even if contains components typical of a Resource Manager and of a Scheduler, has been designed not for replacing these components, but for orchestrate a set of computing systems able to provide physical and virtual resources. In this way we created a unified system, in which the various job flows are managed and optimized.

We evaluated our solution integrating two different well-known IaaS provider proving in practice the interoperability between Cloud and Grid environments.

We carried out four experiments to measure the waiting time of a set of 100 jobs according to the four possible job flows we may have. The results in terms of minimum, maximum and mean waiting time values and the type of job distributions obtained confirmed the goodness of our method and that there is no additional overhead due to our system.

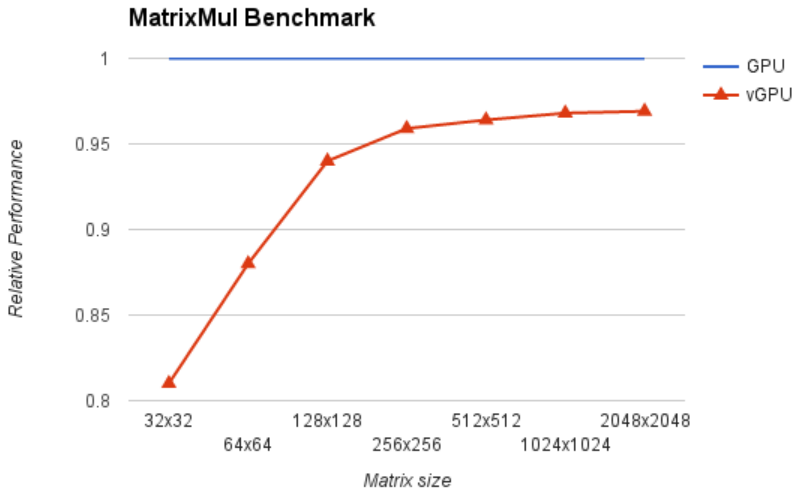


Figure 9: Matrix multiplication benchmark on squared matrix.

Finally as use case, we have also provided an example for the execution of a Grid job on GPU devices via an IaaS provider, which is really important, considering the increasing popularity of the GPGPU approaches in several fields of the sciences of life studies.

The work may be extended in several ways. The *Direct2M* component may be rewritten to remove `ssh` as the command used for accessing servers and Virtual Machines. The DBMS may be removed from the architecture and the system may be re-engineered to be fully distributed adopting for example the protocol 9P described in [34]. Furthermore we can explore innovative approaches for data and big-data management. In this respect, some interesting directions to be taken into consideration are: (*i*) *fragmentation issues* (e.g., [15]); (*ii*) *uncertain data management issues* (e.g., [29]); (*iii*) *general big data management issues* (e.g., [17]).

References

[1] European grid infrastructure. `http://www.egi.eu/`. Accessed on 2015-09-09.

[2] Web site of boto library, a python interface to amazon web services. `https://github.com/boto/boto`. Accessed on 2015-09-09.

[3] Web site of eucalyptus. `http://www.eucalyptus.com`. Accessed on 2015-09-09.

[4] Web site of nimbus project. `http://www.nimbusproject.org/`. Accessed on 2015-09-09.

[5] Web site of the amazon elastic compute cloud (ec2):. `http://aws.amazon.com/ec2/`. Accessed on 2015-09-09.

[6] Web site of the fog library sdk, an interface to openstack cloud environment:. `https://github.com/fog/fog/blob/master/lib/fog/openstack/docs/getting_started.md`. Accessed on 2015-09-09.

[7] Web site of the open cloud computing interface (occi). `http://occi-wg.org/`. Accessed on 2015-09-09.

[8] Web site of the open nebula project. `http://opennebula.org/`. Accessed on 2015-09-09.

[9] G. B. Barone, R. Bifulco, V. Boccia, D. Bottalico, R. Canonico, and L. Carracciuolo. Gaas: Customized grids in the clouds. In *Euro-Par 2012: Parallel Processing Workshops*, pages 577–586. Springer, 2013.

[10] M. Bernaschi, G. Carbone, E. Mastrostefano, and F. Vella. Solutions to the st-connectivity problem using a gpu-based distributed {BFS}. *Journal of Parallel and Distributed Computing*, 76:145 – 153, 2015. Special Issue on Architecture and Algorithms for Irregular Applications.

[11] Stéphane Bressan, Alfredo Cuzzocrea, Panagiotis Karras, Xuesong Lu, and Sadegh Heyrani-Nobari. An effective and efficient parallel approach for random graph generation over gpus. *J. Parallel Distrib. Comput.*, 73(3):303–316, 2013.

[12] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2002.

[13] J. Castillo, ł. Bosque, C. Pedraza, E. Castillo, P. Huerta, and J. I. Martinez. Low cost high performance reconfigurable computing. In W. Vanderbauwhede and K. Benkrid, editors, *High-Performance Computing Using FPGAs*, pages 453–479. Springer New York, 2013.

[14] R. K. Chellappa. "intermediaries in cloud–computing: A new computing paradigm". *INFORMS Meeting*, 1997.

[15] Alfredo Cuzzocrea, Jérôme Darmont, and Hadj Mahboubi. Fragmenting very large XML data warehouses via k-means clustering algorithm. *IJBIDM*, 4(3/4):301–328, 2009.

[16] Alfredo Cuzzocrea, Filippo Furfaro, Giuseppe M. Mazzeo, and Domenico Saccà. A grid framework for approximate aggregate query answering on summarized sensor network readings. In *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops: OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, October 25-29, 2004. Proceedings*, pages 144–153, 2004.

[17] Alfredo Cuzzocrea, Domenico Saccà, and Jeffrey D. Ullman. Big data: a research agenda. In *17th International Database Engineering & Applications Symposium, IDEAS '13, Barcelona, Spain - October 09 - 11, 2013*, pages 198–203, 2013.

[18] A. Dovier, E. Pontelli A. Formisano, and Flavio Vella. Parallel execution of the ASP computation - an investigation on gpus. In *Proceedings of the Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015), Cork, Ireland, August 31 - September 4, 2015.*, 2015.

[19] O. Gervasi F. Vella, I. Neri and S. Tasso. A simulation framework for scheduling performance evaluation on cpu-gpu heterogeneous system. In *Proceedings of the 12th International Conference on Computational Science and Its Applications - Volume Part IV*, ICCSA 2012, pages 457–469, Berlin, Heidelberg, 2012. Springer-Verlag.

[20] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *Berman et al.[2]*, pages 171–197, 2003.

[21] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[22] O. Gervasi, D. Russo, and F. Vella. The aes implantation based on opencl for multi/many core architecture. In *Computational Science and Its Applications (ICCSA), 2010 International Conference on*, pages 129–134, March 2010.

[23] F. Giunta, R. Montella, G. Laccetti, F. Isaila, and F. Blas. A gpu accelerated high performance cloud computing infrastructure for grid computing based virtual environmental laboratory. *Advances in Grid Computing*, pages 121–146, 2011.

[24] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan. Gvim: Gpu-accelerated virtual machines. In *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, HPCVirt '09, pages 17–24, New York, NY, USA, 2009. ACM.

[25] Junjie Hu, Arshad Saleem, Shi You, Lars Nordström, Morten Lind, and Jacob Østergaard. A multi-agent system for distribution grid congestion management with electric vehicles. *Eng. Appl. of AI*, 38:45–58, 2015.

[26] J J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, and F. Silla. An efficient implementation of gpu virtualization in high performance clusters. In *Euro-Par 2009–Parallel Processing Workshops*, pages 385–394. Springer, 2010.

[27] B. Klauer. The convey hybrid-core architecture. In Wim Vanderbauwhede and Khaled Benkrid, editors, *High-Performance Computing Using FPGAs*, pages 431–451. Springer New York, 2013.

[28] M. Mariotti L. Servoli and R. M. Cefalà. A proposal to dynamically manage virtual environments in heterogeneous batch systems. In *IEEE Nuclear Science Symposium Conference Record, 2008. NSS08*, pages 823–826, 2008.

[29] Carson Kai-Sang Leung, Alfredo Cuzzocrea, and Fan Jiang. Discovering frequent patterns from uncertain data streams with time-fading and landmark models. *T. Large-Scale Data- and Knowledge-Centered Systems*, 8:174–196, 2013.

[30] C. Loomis, M. Airaj, M. E. Bégin, E. Floros, S. Kenny, and D. O'Callaghan. Stratuslab cloud distribution. *European Research Activities in Cloud Computing*, page 271, 2012.

[31] Athanasios Naskos, Anastasios Gounaris, and Spyros Sioutas. Cloud elasticity: A survey. In *Algorithmic Aspects of Cloud Computing - First International Workshop, ALGOCLOUD 2015, Patras, Greece, September 14-15, 2015. Revised Selected Papers*, pages 151–167, 2015.

[32] Parallella. Parallella 1-x reference manual, 2014. accessed on July 11, 2015.

[33] S. Venugopal J. Broberg R. Buyya, C. S. Yeo and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[34] K. Thompson H. Trickey R. Pike, D. Presotto et al. Plan 9 from bell labs. In *Proceedings of the summer 1990 UKUUG Conference*, pages 1–9. London, UK, 1990.

[35] E. Ronchieri, D. Cesini, D. D'Agostino, V. Ciaschini, G. Dalla Torre, P. Cozzi, D. Salomoni, A. Clematis, L. Milanesi, and I. Merelli. The wnodes cloud virtualization framework: a macromolecular surface analysis application case study. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 218–222. IEEE, 2014.

[36] A. Merzky S. Jha and G. Fox. Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes. *Concurr. Comput.: Pract. Exper.*, 21(8):1087–1108, June 2009.

[37] L. Shi, H. Chen, J. Sun, and K. Li. vcuda: Gpu-accelerated high-performance computing in virtual machines. *Computers, IEEE Transactions on*, 61(6):804–816, 2012.

[38] B. Sotomayor, K. Keahey, and I. Foster. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th international symposium on High performance distributed computing*, pages 87–96. ACM, 2008.

[39] F. Vella, R. M. Cefala, A. Costantini, O. Gervasi, and C. Tanci. Gpu computing in egi environment using a cloud approach. In *Computational Science and Its Applications (ICCSA), 2011 International Conference on*, pages 150–155. IEEE, 2011.

[40] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, and G. C. Fox. Gpu passthrough performance: A comparison of kvm, xen, vmware esxi, and lxc for cuda and opencl applications. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 636–643. IEEE, 2014.