7

A GPU Accelerated High Performance Cloud Computing Infrastructure for Grid Computing Based Virtual Environmental Laboratory

Giulio Giunta[1], Raffaele Montella,[1] Giuliano Laccetti[2],
Florin Isaila[3] and Javier García Blas[3]
[1]*University of Napoli Parthenope*
[2]*University of Napoli Federico II*
[3]*University of Madrid Carlos III*
[1,2]*Italy*
[3]*Spain*

1. Introduction

Numerical models play a main role in the earth sciences, filling in the gap between experimental and theoretical approach. Nowadays, the computational approach is widely recognized as the complement to the scientific analysis. Meanwhile, the huge amount of observed/modelled data, and the need to store, process, and refine them, often makes the use of high performance parallel computing the only effective solution to ensure the effective usability of numerical applications, as in the field of atmospheric /oceanographic science, where the development of the Earth Simulator supercomputer [65] is just the edge.
Grid Computing [38] is a key technology in all the computational sciences, allowing the use of inhomogeneous and geographically spread computational resources, shared across a virtual laboratory. Moreover, this technology offers several invaluable tools in ensuring security, performance, and availability of the applications. A large amount of simulation models have been successfully developed in the past, but a lot of them are poorly engineered and have been designed following a monolithic programming approach, unsuitable for a distributed computing environment or to be accelerated by GPGPUs [53]. The use of the grid computing technologies is often limited to computer science specialists, because of the complexity of grid itself and of its middleware. Another source of complexity resides on the use of coupled models, as, for example, in the case of atmosphere/sea-wave/ocean dynamics. The grid enabling approach could be hampered by the grid software and hardware infrastructure complexity. In this context, the build-up of a grid-aware virtual laboratory for environmental applications is a topical challenge for computer scientists.
The term “e-Science” is usually referred to computationally enhanced science. With the rise of cloud computing technology and on-demand resource allocation, the meaning of e-Science could straightforwardly change to elastic-Science. The aim of our virtual laboratory is to bridge the gap between the technology push of the high performance cloud computing and the pull of a wide range of scientific experimental applications. It provides generic functionalities supporting a wide class of specific e-Science application environments and

set up an experimental infrastructure for the evaluation of scientific ideas. Our Virtual Laboratory has the ambitious goal to spread elastic and virtually unlimited computing power to scientists belonging to different e-science communities in a transparent and straightforward fashion.
The rest of the chapter is organized as follows: section 2 describes the design and implementation of the application orchestration component. Section 3 is dedicated to the GPU virtualization technique, to the description of key design choices and of some benchmark results, which demonstrate the light impact of our software stack on overall performance; in that section, we also discuss about some high performance cloud computing scenarios depicted by recent gVirtuS enhancements. The need of a high performance file system distributed among virtual machines instances is the main motivation of the section 4; we discuss the design and the implementation of a virtual distributed file system mainly targeted to achieve high performance in dealing with HDF and NetCDF data archives. Our attention on those hierarchical self describing data formats is due to their extensive use in computational environmental science. Hence, section 5 sums up the virtual laboratory software ecosystem and describes the testbed application we have used in order to trim up the implemented features. Finally, the conclusions and future direction in this field are reported in section 6, where the principal guidelines for improving the usability and the availability of the described software components are drown.

2. Contextualization and application orchestration

Recently, a multi-disciplinary effort in virtually all science fields led domain specific scientists and computer scientists to work together in order to obtain from the models the best simulation results in the most efficient and reliable way as demonstrated by the tight coupling of computer science and environmental science in data intensive applications [61].
The growth of the many-core technologies could seem to make obsolete one of the traditional goals of the grid computing technology, i.e. the provision of high performance computing capabilities when local resources are not enough. We feel that this point is not correct and that the increase of computing power will continue to follow the increase of the need for it and vice-versa. This means that large scale computing problems can be managed by aggregating and federating many big computing resources and that grid computing can be the underlying technology for resource federation through a virtual organization approach [42].
Currently, the availability of computing and storage resources is a bottleneck in making progress in many fields of science. This suggests that the computing grids have to be elastic, in the meaning of the ability to allocate, for the needed amount of time, as much computing power as needed by the experiment. The cloud computing technology, which is based on hardware virtualization and many-core technologies, provides a suitable environment to achieve such goals. Grid and cloud computing are strictly related and, in many computer science aspects, share some visions and issues, but while the first focuses on aggregation and federation, the last enables the dynamical hosting of computing and storage resources. In this scenario, one can have “clouds on grids” if the virtualization and the dynamical resource provision is managed by a grid middleware. Otherwise, when grid-computing resources are partially (or totally) dynamically allocated on the cloud, one can have “grids on clouds” [24].
Virtual clusters instanced on cloud infrastructures, aggregated using grid computing middleware and leveraging on virtual organizations, suffer from poor message passing performance between virtual machine instances running on the same real machine [74].

Another source of computing power limitation is the impossibility to access hardware specific accelerating devices as GPUs in a fully transparent way [47].
For these reasons, the development of a grid/cloud-aware virtual laboratory for environmental applications is an interesting and a well-know challenge for computer scientists.

2.1 The grid and cloud enabling approaches

Making an application capable of working on a grid is a complex process called "grid-enabling". This process could be hampered by the grid software and hardware infrastructure complexity, but, above all, it is strictly related to the application that has to be grid-enabled. Applications characterized by an intrinsic parallelism, where the spatially decomposed domain can be processed without any interaction among computing threads appear to be perfectly suitable for the grid. This is the case, for instance, of parameter sweep based algorithms, very common in computational high-energy physics. Apparently, all parallel applications, which rely on tightly coupled concurrent processes, seem to be unsuitable for grid enabling. Numerical models based on systems of differential equations solved by finite difference or finite elements methods belong to this application class. In the world of parallel computing, these applications can achieve high performance thanks to message passing libraries as PVM [67] and, mainly, MPI [46]. In order to migrate MPI concepts in the grid world, MPICH-G2 [52] permits distributed cluster nodes to be aggregated as a single supercomputing resource. Even though this approach is limited by the connection bandwidth among computing nodes, nevertheless it has been successfully used on production environments deployed on Teragrid [35].

2.1.1 The grid-enabling

In the grid lingo, the job is the processing unit in term of piece of software to be executed and producing a result. A grid-enabled application has to be rearranged as a job or pool of jobs, and, moreover, has to be executed in a shared environment, as a guest on a remote machine, ensuring the management of data staging. First generation grid middleware, such as Condor [68] and Globus 1.x and 2.x [39], considered the job as an executable, with specific computing needs, which is dynamically deployed on a target computing element matching the job needs. Typically, data needed by the job are staged-in on the working machine and then the results staged-out. The user must know the requirements of the application in terms of CPU, memory, disk, and architecture. With the rise of web services technology, the next generation of middleware, such as Globus 3.x [40] and Globus 4.x [41], moved towards a different concept of grid enabled application, i.e. the application must be functionally decomposed and each component has to be exposed to the grid leveraging on a wrapping web service. In this case, the user should search for a computing element offering the needed component and low level details about application's needs are hidden to the user. The application is not deployed on-the-fly and can be managed in a safe, secure, and consistent way, and the main effort for grid enabling consists in designing, developing and implementing a suitable web service interface. The complexity introduced by the web service infrastructure and the latency related to that technology is drown back by the flexibility and the semantic coherence of an application made by components interconnected through web services. The fast development of the web service technology, the poor performance in job submission and the complexity in programming and management called

for the last generation of grid middleware Globus 5.x and gLite. Web service technology has been decoupled from authentication, job submission and data management and it is available as a separate tier. A job is executed remotely, using a submission interface, which avoids on-fly deployment.
In our proposed grid infrastructure, we have used the middleware Globus Toolkit version 2.x, 4.x, 5.x (GT2/GT4/GT5), developed within the Globus Alliance and the Open Grid Forum (OGF) with a wide support of institutions belonging to the academia, the government and the business area. The GT4 has been chosen because it exposes its features via web services, using common W3C standards as the Web Service Description Language (WSDL), the Standard Object Access Protocol (SOAP), and the Hyper Text Transfer Protocol (HTTP). More complex features, i.e. service persistence, state and stateless behaviour, event notification, data element management and index services tools, are implemented in a way compliant to that standard. GT4 also supports the GridFTP protocol, an FTP enhanced version, and capable of massive parallel striping and reliable file transfer. Recently, we smoothly migrated under GT5 and explored the interoperability with the EGEE/gLite world.
The application package is the key component of our implementation and is the blueprint of the job execution environment. A framework approach can be used to abstract different application model configurations, by exploiting an object-oriented programming-like methodology. Each application runs in a private custom environment, so that several instances of the same software can be concurrently executed. This approach is based on a repository in which application packages are stored and from where they can be retrieved as instance templates to support a deployment on-the-fly.
A job launcher, invoked as a grid job on a remote machine or locally as well, sets up the virtual private environment and performs the needed stage-in and stage-out operations on the data files. The launching script can be statically installed on the target machine, automatically deployed from the local repository or even staged-in: it represents the job executable file to be run on the target computing-element. This script unpacks the application package, downloaded from a repository or copied from a local directory, unpacks and inflates input data, runs the actual application executable and, finally, deflates results. The framework produces logs based on a XML schema that allows a straightforward parsing.
The resource specification is performed by a description XML file, following the Job Submission Description Language schema [23]. The submitted job is described by specifying the executable path, the current working directory, the files to be staged-in before the execution and staged-out after the job run. All files are named using full-qualified URLs, with protocol details from the target machine. In order to simplify the process of grid enabling and then the application execution as a job, we implemented a custom resource specification pre-processor for the definition of jobs, which relies on an advanced method of labelling and placeholders parsing and evaluation, macro based code explosion and late binding capabilities.

2.1.2 The cloud-enabling

While the grid-enabling process is well defined, the concept of cloud enabling is still under drawing. NIST defines cloud computing as a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources that can be provisioned and released with minimal management effort or service

provider interaction [55]. Within the cloud paradigm, server time and network storage can be provisioned to the user in an on-demand self-service fashion without requiring human interaction. Resources are made available over the network and ubiquitously accessed through standard mechanisms such as thin or thick client platforms. According to current demand, computing resources (both physical and virtual) are dynamically assigned and reassigned to serve all users that generally have no control or knowledge over the exact location of the provided resources. The computing resources can be elastically provisioned to scale up and released to scale down: the set of resources appears to be infinite (infinitive computing) [25].

Any kind of software, including operating systems and applications can be deployed and then run on the best matching computing resource that is provided for rent to the user. The user doesn't manage the underlying cloud infrastructure, but has control over operating systems, storage and applications. This is the so-called Infrastructure as a Service (IaaS) and could be considered as the lowest level in a cloud oriented infrastructure stack [33]. The next level is the Platform as a Service (PaaS, where the user has control over the deployed applications and possibly the application hosting environment [71]. The highest level of the stack is the Software as a Service (SaaS): one can use the provider's applications running on a cloud infrastructure from various client devices through a client interface such as a Web browser (e.g., web-based email). The user can modify only some application configuration parameters, but doesn't manage or control the underlying cloud infrastructure [54].

If the cloud infrastructure is owned by a single organization and is used only inside that organization, then it is a "private cloud". A cloud is a "community cloud" when the infrastructure is shared by several organizations. In e-science applications, "hybrid clouds" are becoming popular.

In our context, resources include storage, processing, memory, network bandwidth, and virtual machines. Since a virtual laboratory is designed to serve the world of computational science, a hybrid cloud solution appears to be the most suitable grid deployment. Cloud-enabled software has to take full advantage of the cloud paradigm by being service oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability or encapsulated in a virtual machine image and executed as a virtual machine instance.

Experience in grid enabling application may be useful in order to design and implement cloud enabling guidelines, especially concerning the virtual laboratory software components. An object-oriented approach, yet followed in grid enabling, may be replicated in this more complex scenario. The process could be divided in two main steps. First, the application package is deployed in a virtual machine image and then, once the virtual machine is instanced, the application performs its lifecycle. The deployment of an application on a virtual machine image could be a complex task if the main goal is the cloud infrastructure best resource allocation. We have developed a software tool specifically designed to help and partially automate the creation of application appliances. Our Virtual Machine Compiler (VMc) produces a XML description of the needed virtual machine in order to run a generic piece of software in a virtualized environment with minimal resources for achieving a required performance. Once installed and configured by the user, the application is executed in a full featured virtual machine under the control of the VMc, that evaluates the disk and memory needs and, above all, the needed services and libraries. The application lifecycle is managed by an abstract framework in which the user can implement software components performing specific actions, such as answering to events fired when the virtual machine is up and running, or before the application is going to be executed (in order to implement data

staging-in), or during the application run (for example for progress status notification), or even when the application finished (for the staging-out of the results) and the virtual machine instance is ready to be shouted down. The VMc output is a XML file which contains the software and hardware requirements needed by the application. This output can be converted to platform-specific installation/configuration file (as the Fedora/RedHat/CentOS kickstart file). The VMc output will be used in the virtual machine image creation using commonly available open source software packages. Finally, the application environment is cloned on the virtual machine image and made ready to be executed within a software appliance. This process is far from being completely automatic, but, once performed successfully, it is no more in charge to the user. The computational scientist has just to instantiate the virtual machine and wait for results. In case of distributed memory parallel programs, the tool helps to automatically set up two virtual machine images, one for the head node and the other for the computing node, and all the other needed features, such as the shared file system, user authentication and message passing interface libraries.

2.2 Application orchestration

In many scientific applications, the final result is obtained by assembling different components executed as jobs on remote machines as workflows [70]. Each component could be related to its previous/next component as data producer or data consumer, defining a so-called computational pipeline, with one job for one component.

For example, let us consider this simple application:

1. A regional-scale atmospheric model waits until data can be downloaded from a specified service.
2. It acquires the boundary and initial conditions from a global-scale forecast, and runs for a specified time period.
3. When data are ready to be processed, another job, which simulates an ocean circulation model, uses atmospheric data as boundary conditions.
4. When this second job finishes its computing task, the produced data are consumed by another job which simulates the wind driven sea wave propagation and forecasts the wave height/period and their direction fields.
5. At last, the user retrieves all pipelined outputs produced by all models.

This scientific application could be implemented via shell scripts and middleware specific job submission tools, specifying the target machine in the script.

The Globus Toolkit grid middleware provides job submission tools via web services (4.x), pre-web services (2.x) and web services-independent (5.x) infrastructure without any support for job flow scheduling and resource broking. Other grid technologies offer a full support of direct acyclic graph job workflow with conditional branches, recovery feature and graphic user interfaces. Our custom software solution provides domain scientists with a full configurable grid and cloud-computing tool, which minimizes the impact of the computing infrastructure.

In a hybrid grid/cloud infrastructure scenario, the shell script instantiates virtual machines, without concerning about where to execute the computation but the needed computing resources.

This approach, though operatively correct, presents many disadvantages. The user must know many details about the script programming language, the job submission, the system environment setup, and, if the computing resources are virtualized, even about the virtual machine lifecycle and the interaction with instances. The developed code is tightly coupled

to its application and any change to the job behaviour or model configuration affects the entire application. In case of complex job fluxes, like in a concurrent ramification context, for example when a weather simulation model forces both wave propagation and oceanic circulation models, the control code may grow in complexity and data consuming/production relationships could be hard to implement, since several synchronization issues arise. Moreover, this kind of approach is potentially insecure because the user must be logged into the system to run a script, and this scenario is not applicable in the case of an interactive application on a web portal.

2.2.1 Orchestration components design and implementation

The orchestration component design has been driven by the goal of disclosing to the final user the power of a hybrid grid/cloud infrastructure and hiding technical details.
The virtual laboratory main component is the Processing Unit (PU). The PU may be seen as a black box feed by input data, performing a computation task and producing output data. In a grid-computing environment, the PU could be considered as a job submitted to a grid machine matching the computational needs. In a dynamical infrastructure as a service cloud, the PU can be totally embedded in a virtual machine image and deployed to be executed using the requested computing resources. To the user of the virtual laboratory, the view has completely changed: the cloud elasticity permits to consider computing resources as potentially infinite. In our architecture stack, the PU represents the highest level of interaction component and leverages on the cloud management software components.

2.2.2 JaClouX, a cloud independent Java API

Different cloud infrastructure systems, such as Amazon AWS, Open Nebula, Open Stack, Eucalyptus, etc, offer similar services through different web service interfaces, so a cloud-aware component is needed in order to manage virtual machine instances on different clouds. JaClouX (Java bindings for Cloud eXtensions) [9] is a unified application program interface to the cloud written in Java and it represents a fundamental change in the way clouds are managed, breaking the barriers among proprietary, closed clouds and open scientific clouds. JaClouX is our cloud management component; it is independent of the underlying cloud software and interacts with other components as Images, Instances, Locations, and Sizes. Moreover, JaClouX is cloud-aware thanks to a plug in architecture. The support to any cloud software can be added by just implementing a new Driver component. Respect to other similar APIs, such as libcloud [13] and JClouds [8], JaClouX provides some enhanced specific features for storage and high performance computing.
While most cloud software manage virtual machine images and instances in a quite similar way, the storage and other high level services turn out to be different among the various cloud solutions. For example, AWS offers the Simple Storage Service (S3) implemented in Eucalyptus as Walrus. The Eucalyptus storage controller provides a service similar to the AWS Elastic Block Storage (EBS). The OpenNebula stack does not provide S3-like services, but leverages on shared network file systems on which the virtual machine instances are attached. JaClouX's Driver component provides an abstraction of both S3 and EBS using the Simple Storage Component (SSC) and the Elastic Block Component (EBC). The Driver component can provide the missing cloud features by software emulation.
The Virtual Computing Cluster (VCC) component permits the dynamical creation of high performance computing cluster on-demand. A VCC is defined specifying the virtual machine image for the head and the compute nodes and the number of working nodes

instances. VCC manages the user authentication and the execution authorization in a coherent fashion deploying automatically, if needed, the grid security infrastructure components. The virtual network file system permits to the head node user to execute on working nodes using secure shell and perform some other operation in the distributed memory parallel programming ecosystem. VCC takes in charge file system related issues abstracting the low level implementation especially regarding the distributed parallel file system. Last but not the least, VCC interacts with the cloud scheduler, where possible, in order to locate the virtual computing nodes in the most effective and high performance way: i. e. groups of computing nodes belonging to the same virtual cluster can be allocated on the same real machine, so MPI optimization for virtual networks can be used in the best way; in a similar way, virtual working nodes needing for GPGPU acceleration support are spawn on the real hardware achieving the best possible computing effort.

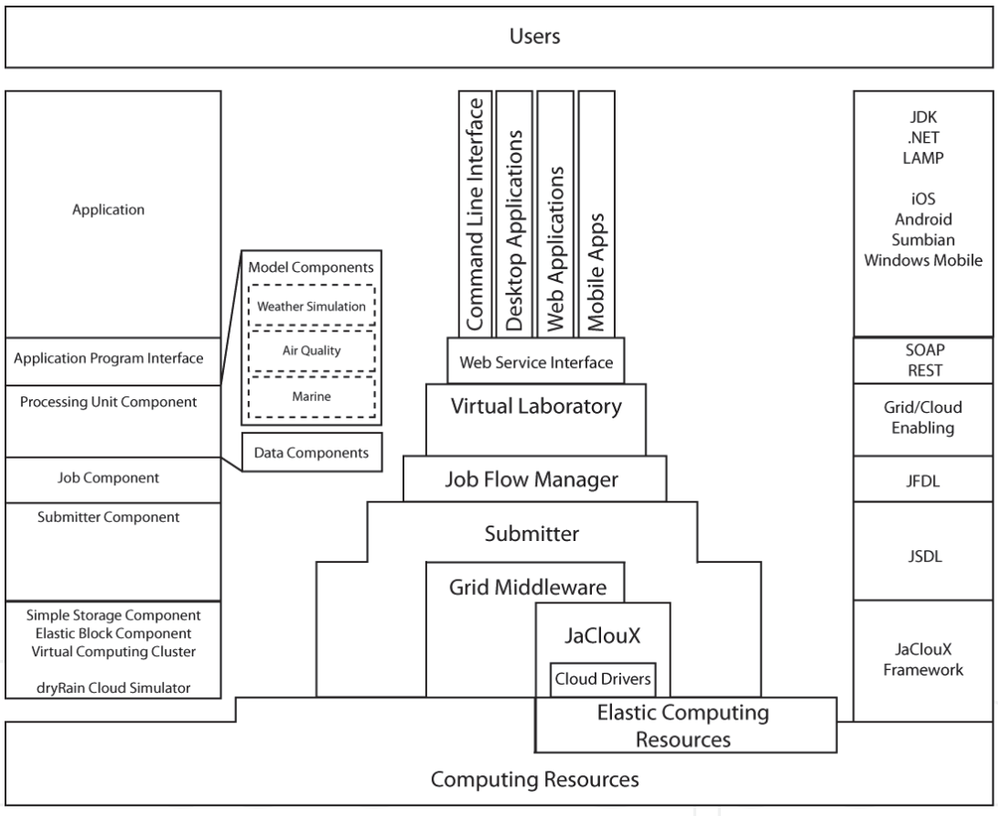


Fig. 1. The Virtual Laboratory big picture. On the right the component stack schema. In the centre area the component interactions/dependences diagram. On the right the main technologies used each application tier.

2.2.3 Managing the application workflow

The automation of scientific processes in which tasks are structured based on their control and data dependencies are defined as *scientific workflows*. Scientific applications on Grids gain several advantages following the workflow paradigm. The ability to build dynamic applications, which orchestrate distributed resources using resources that are located in a particular domain, accelerates the advance in knowledge in many scientific fields. The

workflows permit to increase the throughput or reduce execution costs spanning the execution of scientific software on multiple administrative domains to obtain specific processing capabilities. The highest advantage in the workflow technology application in e-Science is in the integration of multiple teams involved in management of different parts of the experiment promoting inter and cross organization collaborations [48]. With the advent of the cloud technology applied to computational sciences the potential importance of scientific workflows increased: the scientist is in the power of allocate all the needed computing power doesn't caring about the effective availability.

In order to enhance flexibility and to minimize the impact on the grid configuration, we have implemented a software component for scientific workflow management. The Job Flow Manager (JFM) component is an evolution of the previously developed Job Flow Scheduler [26] improved with the cloud computing and the on demand virtual computing cluster support. The JFM component plays a key role in the grid application orchestration. Using this tool the entire complex, multi branch, grid application can be configured through a XML file. JFM takes care of submitting jobs to computing nodes without concerning whether the target machine is a real computing resource or a virtual computing node dynamically allocated on demand on a cloud. The JFM have been designed in a modular fashion and does not depend on the actual grid middleware or cloud ecosystem.

The main JFM role is the submission of jobs or virtual machine instances creation to the propter computing resource abstracting the Processing Unit in our virtual laboratory component stack. The Submitter Component (SC) is a piece of software dealing with the grid middleware and/or with the cloud infrastructure interface. As previously stated, the jobs are formally described with an extension of the JSDL, providing syntactic and semantic tools for job description in terms of virtual machines instances. The user may create virtual computing resources and instantiate them when needed. We defined a XML schema, called Job Flow Definition Language [59] so one can define virtual laboratory experiments as XML files. Thanks to the interaction between the JFM and the SC components, the submitted job could face as a classical grid job wrapped an executable on a remote machine or an application appliance to instantiate, run and destroy on a cloud computing provided resource. The SC is the software interface connecting the PU with the computing resource. We have designed the Submitter component abstracting virtually any kind of remote execution technology including HPC, grid or cloud (via JaClouX).

In the virtual laboratory an experiment is an aggregated of Processing Units, each of them, connected with the others in a direct acyclic graph, draws the computing steps as the nodes of the experiment graph.

The Virtual Lab and the Job components (VLC, JC) represent our playground for scientific workflows form an higher point of view and interacts with the user interface (VLC) and with Submitter component (JC). In particular, the Job component takes in charge the list of the jobs to be completed before the actual processing unit is executed and the list of jobs to be submitted when the PU work eventually ends up.

Finally, VLC provides a friendly user interface through a desktop client, a mobile app, an interactive or batch console or even a web application (Figure 1).

3. GPGPU virtualization

Recently, scientific computing has experienced on general purpose graphics processing units using massive multi core processors available on graphics devices in order to

accelerate data parallel computing tasks. One of the most successful GPU based acceleration system is provided by nVIDIA and relies on the CUDA programming paradigm and tools [16]. Currently, in the world of high performance computing clouds, virtualization does not allow the use of accelerators as CUDA based GPUs in a transparent way. This happens because of the communication between virtual and real machines, of the communication between guest and host on the real machine side, and of the issues related to the vendor specific interface (on the virtual machine side). The use of the GPGPUs massive parallel architecture in scientific computing is still relegated to HPC clusters. A serious limitations exist on the overall potential performances of an on-demand instanced high performances computing cluster and, in general, on the use of cloud computing infrastructure as an high performance computing system based on elastically allocated resources.

3.1 gVirtus architecture and design

gVirtuS (GPU Virtualization Service) is our result in GPGPUs transparent virtualization targeting mainly the use of nVIDIA CUDA based accelerator boards through virtual machines instanced to accelerate scientific computations [45].

The basic virtualization idea is based on a split driver approach [37] in which a front-end is deployed on the virtual machine image and a back-end is hosted on the real machine. The communication technology between the front and the back-ends is a key component in order to achieve high performances. The front-end and the back-end layers implement the uncoupling between the hypervisor (which has in charge the control of the acceleration device) and the communication layer. A key property of the proposed system is its ability to execute CUDA kernels with an overall performance similar to that obtained by real machines with direct access to accelerators. This has been achieved by developing a component that provides a high performance communication between virtual machines and their host, and implements an efficient data transfer between the guest VM's application, which use the kernel, and the GPU, which runs it.

On the front-end side, the component packs the API invocation, recovers the result values and automatically translates host and device memory pointers. This approach permits to enforce the isolation overcoming some hardware-related specific limitations. On the front-end a CUDA library stub imitates the real CUDA library implementation interacting with the back-end in a transparent way. This architectural scheme makes evident an important design choice: the GPU virtualization is independent of the hypervisor.

The approach can naturally be extended to a wide range of devices designed to work in a non-virtualized environment, but the overhead introduced by the system can be prohibitive for HPC applications without an efficient and reliable high performance communication system between guest and host machines. Due to the potentially large size of the input/output data of a CUDA kernel, this is particularly relevant for the GPU accelerators that span many pages of contiguous memory. Our effort was mainly aimed at delivering a GPU virtualization suitable for the performance requirements of current HPC scientific applications and to develop CUDA API as much as possible hypervisor and communicator independent, but the hypervisor proprietary virtual/real machine communication technology affects deeply the over all performances.

In the world of virtualization and cloud computing some hypervisors reached the status of standard *de facto*:

- Xen [21] is a hypervisor that runs directly on the top of the hardware through a custom Linux kernel. Xen provides a communication library between guest and host machines,

called XenLoop, which implements a low latency and wide bandwidth TCP/IP and UDP connection both among virtual machines running on the same host and virtual machine and host machines. XenLoop can reduce latency up to a factor of 5 and can increase efficiency up to a factor of 6. Moreover, it is application transparent and implements an automatic discovery of the supported virtual machines [73]. We excluded Xen from our test-bed because there is no official support of nVIDIA driver for this hypervisor.

- VMWare [19] is a commercial hypervisor running at the application level. VMWare provides a high performance communication channel between guest and host machines, called VMCI (Virtual Machine Communication Interface), leveraging on a shared memory approach. VMWare offers a datagram API to exchange small messages, a shared memory API to share data, an access control API to control which resources a virtual machine can access and a discovery service for publishing and retrieving resources [20].
- KVM (Kernel-based virtual machine) [10] is a Linux loadable kernel module now embedded as a standard component in most of Linux distributions, with a full virtualization support. KVM offers a high performance guest/host communication component called vmChannel. This component exposes a set of fully emulated inter virtual machines and virtual/real machine serial ports and network sockets, based on a shared memory approach. Unfortunately, most of the promising vmChannel features are not yet fully implemented [11].

The use of the TCP/IP stack to permit the communication between virtual machines and a real machine is common feature in virtualization. This approach is not only hypervisor independent, but deployment independent too: the virtual resource consumer and the real producer can be on physical different computing elements. This is a cool feature, especially if in the field of cloud computing, but the introduced overhead is unacceptable for HPC applications. Nevertheless the use of TCP/IP could be interesting in some deployment scenarios where the size of the computing problem, the network performances and, above all, the GPU acceleration justify the general overhead.

In order to be hypervisor and communication technology independent, we designed a fully pluggable. It is an independent communication channel and allows the communicator change through a simple configuration file. Moreover, if a new kind of communicator is implemented, it can be plugged in without any change in the front-end and the back-end layer.

CUDA accelerated applications interact with the front-end CUDA library in a standardized manner. The CUDA library instead of dealing directly with the real nVIDIA hardware interacts with our GPU virtualization front-end. The front-end, using the communicator component, packs the library function invocation and sends it to the back-end. The back-end deals with the real nVIDIA hardware using the CUDA driver; it unpacks the library function invocation and suitably maps memory pointers. Then it executes the CUDA operation, retrieves the results and sends them to the front-end using the communicator. Finally, the front-end interacts with the CUDA library by terminating the GPU operation and providing results to the calling program. Any accesses to the GPU is routed via the front-end/back-end layers under control of a management component, and data are moved from GPU to guest VM application, and vice versa.

This design is hypervisor independent, communicator independent and even accelerator independent, since the same approach could be followed to implement different kinds of

virtualization. The platform is designed to emulate future heterogeneous many-core chips comprised of both general and special purpose processors (Figure 2).

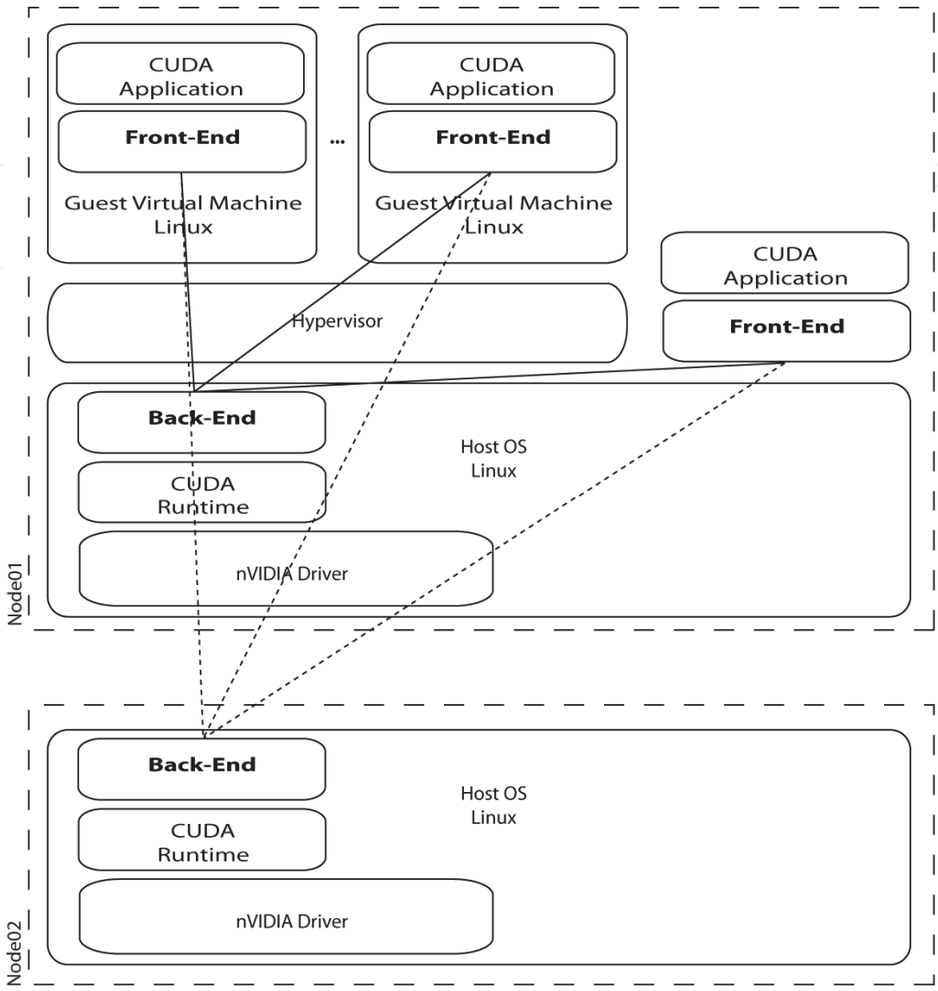


Fig. 2. The gVirtuS schema. In the single node deployment scenario the CUDA application can interact with the back-end with, or without, the hypervisor interaction. If the back-end is on a different computing node (cluster deployment scenario), then the application can be accelerated by GPGPUs devices physically installed on different machines. The actual performances (and benefits) depend of the application, the problem size and the network bandwidth.

3.2 gVirtus implementation

The implementation, in C++, is related to an x86-based multi-core hardware platform with multiple accelerators attached via PCIe devices, running Linux as both host and guest operating system. According to the underlying idea of high performance cloud computing applications, we implemented the virtual accelerator in a hypervisor independent fashion and in a fully configurable way. In particular, we focused on VMware and KVM hypervisors. The GPU is attached to the host system and must run its drivers at a privileged level that can directly access the hardware. Memory management and communication

methods for efficient sharing of the GPU by multiple guest VMs, have to be implemented at the same run level.

We implemented a wrapper stub CUDA library with the same interface of the nVIDIA CUDA library. Our library performs some processing and passes parameters to the lower level components. The library currently implements all function calls synchronously and is able to manage multiple application threads. Since the source code of both library and driver are strictly closed, the wrapper library running in the guest virtual machine intercepts CUDA calls made by an application, collects arguments, packs them into a CUDA call packet, and finally sends the packet to the front end. The front-end driver manages the connections between the guest virtual machine and the host machine. It uses the services offered by the communicator component to establish event-based channels through both sides, receiving call packets from the wrap CUDA library, sending these requests to the back end for execution over a shared call buffer and relaying responses back to the wrap CUDA library.

We designed and implemented a component to achieve a high performance communicator working with the KVM hypervisor: vmSocket. This component exposes Unix Sockets on virtual machine instances thanks to a QEMU device connected to the virtual PCI bus. The device performs register based I/O operations using dynamically allocated memory buffers.

We focused on concentrating most components on the guest machine side, in order to make reusable as many components as possible. On the host side, the back end mediates all accesses to the GPU and it is responsible for executing the CUDA calls received from the front end and for returning the computed results. Once a call has been executed, it notifies the guest and passes the results through the connector component. The back end has been implemented as a user-level module to avoid the additional runtime overhead of multiple accesses to the user space CUDA memory. This component converts the call packets received from the front end into CUDA function calls; notice that it is the wrap CUDA library host side counterpart. Finally, the nVIDIA CUDA library and the driver interact with the real device.

GPU based high performance computing applications usually require massive transfer of data between host (CPU) and device (GPU) memory. In a non-virtualized environment, a memory copy operation invoked by a CUDA application copies data between host memory and GPU memory, managed by the nVIDIA driver. This can be avoided by making a call to the CUDA API enabling the allocation of host-mapped GPU memory, and implies zero-copy of data. In a virtualized environment the virtual machine isolation adds another layer between the CUDA application working in the guest memory and the CUDA device.

In our implementation there is no mapping between guest memory and device memory because the gVirtuS backend acts in behalf of the guest running virtual machine, then device memory pointers are valid in the host real machine and in the guest as well. In this way the front end and the back end sides interacts in a effective and efficient way because the memory device pointers are never de-referenced on the host side of the CUDA enabled software (with gVirtuS the host side is the guest virtual machine side): CUDA kernels are executed on our backend side where the pointers are fully consistent. Exists a sort of pointers mapping between backend and frontend sides managed by the backend due to support the execution of CUDA function via the vmSocket channel. When a CUDA function have to be invoked on the device, the stub CUDA library interact with the front end. The front end prepares a package containing the name of the function to be invoked and the related parameters. The front end/back end communication handles three kinds of

parameters: automatic scalar variables, host memory pointers and device memory pointers. The serialization function packs the scalar variables without any modification, uses the value pointed by the pointer for each host memory pointer and considers as unsigned 64 bits long integer each device memory pointer. The de-serialization function works in an analogue, but opposite way when the communication is from the backend to the frontend.
Sharing memory between virtual and real machines could be a strange feature in the world of cloud computing where the isolation among VM instances and computing resource is a primary goal. Nevertheless in the high performance cloud computing context this feature could be extremely limiting: using shared memory a MPI communication channel could implement transparently a latency free network among virtual computing nodes without any change to the distributed memory parallel software. In order to achieve better performances in our GPU virtualization we designed and implemented the VMShm component. VMShm ensures high performance memory block copying between guest and host system using an implementation approach similar to vmSocket via the PCI-e virtual driver.
We compared and contrasted many hypervisor/communicator configurations (VMWare: VMCI/tcp, KVM-QEMU: vmSocket/tcp) and, using standard CUDA SDK benchmark programs, demonstrated our GPU virtualization (KVM-QEMU with vmSocke) achieves performances not so much different than the GPU with no virtualization [45].

3.3 Cloud deployment

As mentioned above, we have designed and implemented the gVirtuS GPU virtualization system with the primary goal of using GPUs in a private computing cloud for high performance scientific computing. We have set up a department prototypal cloud computing system leveraging on the Eucalyptus open source software. Eucalyptus, developed at the University of Santa Barbara in California [], implements an Infrastructure as a Service (IaaS), as it is commonly referred to. The system gives users the ability to run and control entire virtual machine instances deployed across a variety of physical resources in the same way the Amazon Web Service infrastructure works [1]. Eucalyptus enables users familiar with existing Grid and HPC systems to explore new cloud computing functionalities while maintaining access to existing, familiar application development software and Grid middleware. In particular we were interested in setting up high performance computing clusters on-demand, leasing the resources upon an existing infrastructure [72]. In our test aimed at assessing the performance of gVirtuS in a high performance computing cloud system, we have used an Intel based 12 computing nodes cluster, where each node is equipped with a quad core 64 bit CPU and an nVIDIA GeForce GT 9400 video card with 16 CUDA cores and a memory of 1 Gbyte.
We carried out an experiment focused on the gVirtuS behaviour in a private cloud, where the GPUs are seen as virtual computing nodes building a virtual cluster dynamically deployed on a private cloud [66]. We developed an *ad-hoc* benchmark software implementing a matrix multiplication algorithms. This software uses a classic memory distributed parallel approach. The first matrix is distributed by rows; the second one by columns, and each process have to perform a local matrix multiplication. MPICH 2[] is message-passing interface among processes, whereas each process uses the CUDA library to perform the local matrix multiplication.
The evaluation process results show that the gVirtuS GPU virtualization and the related sharing system allow an effective exploitation of the computing power of the GPUs [45]. We

note that without such component the GPUs could not be seen by the virtual machine and it would be not possible to run this experiment on a public or private cloud hosting an dynamic (on demand) virtual cluster using, for example, JaClouX features.

4. High performance parallel I/O

The needs of scientific applications have driven a continuous increase in scale and capability of leading parallel systems [7]. However, the improvement in rates of computation has not been matched by an increase in I/O capabilities. For example, earlier supercomputers maintained a ratio of 1GBps of parallel I/O bandwidth for every TFLOP, whereas in current systems 1GBps for every 10 TFLOPS [22] is the norm. This increased disparity makes it even more critical that the I/O subsystem is used in the most efficient manner. Scalable I/O has been already identified as a critical issue for PFLOP systems. Future exascale systems forecasted for 2018-2020 will presumably have O(1B) cores and will be hierarchical in both platform and algorithms [17]. This hierarchy will imply a longer path in moving data from cores to storage and vice-versa, resulting in even higher I/O latencies, relative to the rates of computation and communication in the system. The GPGPU-based systems introduce an additional level into this hierarchy, making a high-performance low-latency solution for file accesses even more necessary.
An additional contribution of this chapter is to extend our previous work on designing and implementing multiple level parallel I/O architectures for clusters [50] and supercomputers [51] to virtualized GPGPU-based cloud architecture in order to improve the file access performance (I/O latency and throughput) of parallel applications.
Our proposed solution is based on a multi-tier hierarchy as depicted in Figure 3, the parallel I/O architecture is organized on six tiers: application, application I/O forwarding, client-side cache, aggregator I/O forwarding, I/O node-side cache and storage. In the figure we show the components of the parallel I/O architecture and on the margins, we depict how the logical components map on physical nodes.

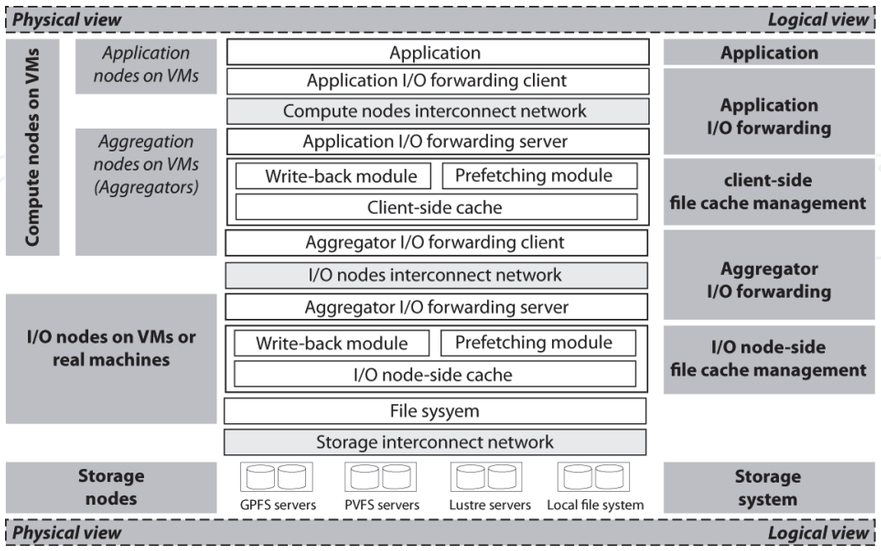


Fig. 3. Parallel I/O architecture organized on six logical tiers: application, application I/O forwarding, client-side cache, aggregator I/O forwarding, I/O node-side cache and storage.

4.1 Application tier.

From a parallel I/O perspective applications run on virtual machines and access the file systems through file interfaces such as MPI-IO or POSIX. As described in section, the CUDA calls of the applications running on virtual machines are intercepted in forwarded in zero-copy fashion to the guest machine where they are processed by the CUDA run time. The transfers between main memory and GPGPU memory as well as GPGPU processing can be completely overlapped by completely overlapped with the service of file system calls.

4.2 Application I/O forwarding tier.

Application I/O forwarding tier resides on all application nodes and has the goal of forwarding the file accesses to the next tier through the compute nodes interconnect network. The forwarding is done on-demand, when the application issues the file access.
This forwarding tier leverages VMSocket and VMShm in order to achieve high performance in intra virtual machines interactions.

4.3 Client-side file cache management tier.

The client-side file cache module resides on the memory of virtual machines and has the role of aggregating small file access requests from several application processes running on virtual machines and performing the file access on behalf of them. The application accesses the client-side cache through the application I/O forwarding tier. i.e. through VMSocket. The objective of client-side file cache module is to provide the management of a file cache close to the applications and to offer efficient transfer methods between the applications and I/O subsystem. The main tasks of the module are buffer management and asynchronous data staging, including write-back and prefetching modules. This module is generic, is network- and file system-independent and is common for both cluster and supercomputer architectures. The client-side cache management tier absorbs the writes of the applications and hides the latency of accessing the I/O nodes over interconnect networks.

4.4 Aggregator I/O forwarding tier.

Aggregator I/O forwarding tier resides on all virtual machines and has the goal of forwarding accesses at file block granularity to the I/O nodes through vmSocket library. The communication with the compute nodes are decoupled from the file system access, allowing for a full overlap of the two operations.

4.5 I/O node-side file cache management tier.

The I/O node-side file cache management tier resides on all I/O nodes. The I/O nodes can run either on virtual machines or on a real machine. Running I/O nodes on virtual machines bring benefit when the load is highly variable and starting or shutting down virtual machines can contribute to adapt the cost of operation of the infrastructure. The objective of I/O node-side file cache module is to provide the management of a file cache close to the storage and offer efficient transfer methods between virtual machines and the storage system. The I/O node-side file cache management tier absorbs the blocks transferred asynchronously from the client-side cache through the aggregator I/O forwarding client, and hides the transfers between the I/O nodes and file systems. An I/O thread is responsible for asynchronously accessing the file systems by enforcing write-back and prefetching policies.

4.6 Storage system tier.

The file system components run on dedicated file servers connected to storage nodes through the storage interconnection network. The storage system provides the persistent storage service and can be any file system providing a VFS (virtual file system) interface. The I/O node-side file cache management tier provides an agile and elastic proxy layer that allows the performance of the file systems to scale up with the load generated by the applications.

4.7 Data staging.

The architecture presented in the previous sections scales both with the computation and I/O requirements of the applications but at the cost of a hierarchical organization of file caches. In order to make an efficient use of hierarchy of file caches our solution employs a multiple level data staging. The first file level caching is deployed on virtual machines of the applications and have the role improving the performance of the accesses to the file systems by overlapping the GPGPU and processor cores computation with I/O transfers between computational virtual machines and I/O nodes. The I/O servers manage the second level of distributed caching at I/O nodes. File blocks are mapped to I/O servers and each server is responsible for transferring its blocks to and from the persistent storage. In order to hide the latency of file accesses, data staging acts in coordination both on virtual machines and on I/O nodes. The data staging consists of two flows corresponding to the write and read operations.

4.7.1 Multiple-level write-back.

After a file write is issued on a virtual machine, data are pipelined from compute nodes through the I/O nodes to the file systems. The application I/O forwarding tier transfers the application write request to the client-side cache management tier. The cached file blocks are marked dirty, the application is acknowledged the successful transfer and is allowed to continue. The responsibility of flushing the data from client-side cache to I/O node over the I/O network belongs to a write-back module on a virtual machine. On the I/O node a write-back module is in charge of caching the file blocks received from the compute nodes and flushing them to the file system over the storage network. The write-back policies are based on a high/low water-mark for dirty blocks. The high and low water marks are percentages of dirty blocks. The flushing of dirty blocks is activated when a high water mark of blocks is reached. Once activated the flushing continues until the number of dirty blocks falls below a low water mark. Blocks are chosen to be flushed in the Least Recently Modified (LRM) order. In order to efficiently hide the latency, coordination along the pipeline is necessary. In a previous work we made an extensive evaluation of coordination strategies for Blue Gene supercomputers [51]. The evaluation of this coordination for GPGPU clouds is subject of current research.

4.7.2 Multi-level prefetching.

The file read pipeline involves transfers from storage through storage nodes and I/O nodes toward the compute nodes. Our prefetching solution for GPGPU clouds proposes two prefetching modules on the computing virtual machines and I/O nodes, each of which enforcing its own prefetching policy. The prefetching mechanism is leveraged in both cases by an I/O thread.

Our prefeching solution leverages the characteristics of the stream processing model of the GPGPU applications [32]. In GPGPU applications the kernel operations define unambiguously both the input and output data. We exploit this information for issuing early prefetching requests on the computing virtual machines. This approach activates the prefetching pipeline in our architecture and allows the overlapping of GPGPU computing and prefetching for future computations. The evaluation of this solution is subject of current research.

5. The virtual laboratory

A domain scientist can use the Virtual Laboratory through a desktop, web or command line client application. An experiment is designed by connecting processing units in a data producer/consumer schema represented as a direct acyclic graph. The scientist can configure and run his experiment selecting and assembling each component from a palette or submitting a JFDL file to the Virtual Laboratory Engine.
Our virtual laboratory is specialized (but not limited to) environmental modelling experiments and applications. Starting in 2003, we run operationally a weather and marine forecast grid application integrating weather, sea wave propagation, ocean circulation and air quality models and offering products, focused on the Bay of Naples (Italy) reaching a nested ground resolution of about one kilometre. Common citizens, via a web portal interface or smart-phones apps, consume the application products. We have used this application as test bed to evolve our laboratory till the actual state of the art of a fully elastic-Science approach.
Relying on our grid-enabled components, we implemented real world applications based on the close integration between environmental models, environmental data acquisition instruments and data distribution systems glued by the hybrid grid/cloud computing technology.

5.1 Laboratory's virtual components

The virtual laboratory is designed in order to provide computing support to any kind of experiment deployable as a scientific workflow. Our packaged framework for grid enabling legacy software components simplifies the process of including additional grid components. In the current implementation, the processing units black-boxing the computing models are based on the cloud-enabling scheme. In this way, distributed memory parallel models, i.e. weather models, are executed on a dynamically created computing cluster or on a real HPC cluster as well, just changing an experiment parameter. As previously stated, we focused our interest in environmental modelling, grid and cloud enabling the needed components for all around coupled forecasts, simulations and scenario analysis: weather, air quality, ocean currents and sea waves models. Over the last decade, we implemented several environmental models. Some of them are currently used in our virtual laboratory production; others have been substituted with more up to date components.

5.2 Weather simulation components

Weather models are the main driver in this kind of experiments and applications, so we grid enabled the MM5 (Mesoscale Model 5) [56] and the WRF (Weather and Research Forecasting model) [57], the latter beingcurrently our first source of operational weather forecasts. It is a mesoscale numerical weather prediction system designed to serve both operational

forecasting and atmospheric research needs. It features multiple dynamical cores, a 3-dimensional variational data assimilation system, and a software architecture allowing for computational parallelism and system extensibility. WRF provides to operational forecasting a model that is flexible and efficient computationally, while offering the advances in physics, numerical solution of equation and data assimilation contributed by the research community. WRF allows researchers the ability to conduct simulations reflecting either real data or idealized configurations. WRF is suitable for a broad spectrum of applications across scales ranging from meters to thousands of kilometres. WRF is modular and its components could be parted in pre-processing, computing and post-processing units.
We grid enabled WRF in order to run each model anywhere on the grid and/or even on the cloud. The components available on our virtual laboratory wrap the Geogrid for terrain preparation, the Ungrib for initial and boundary condition processing and the Metgrid for initial data interpolation over spatial domains. This components could be run on serial or parallel target computing elements, while the Real component, for input data preparation, and the Wrf component, i.e. the model itself, have to be run on parallel computing elements. In particular, Wrf (version 3.2 and beyond) could be executed using a GPU equipped parallel computing cluster dynamically instanced on a cloud resource supporting a gVirtuS based GPGPU virtualization. The post-processing components wrap over standard ARW-POST features and custom developed coupling software.

5.3 Air quality components

Air quality models permit to produce air pollution forecasts maps and have to be fed by emission models of primary chemical pollutants. Currently, we use CHIMERE [29] which integrates both models. It is a multi-scale model primarily designed to produce forecasts of the concentration of ozone, aerosols and other pollutants. The model performs long-term simulations, entire seasons or years, for emission control scenarios and what-if hypothesis studies. CHIMERE runs over meso to urban scale making possible advanced simulation analysis thanks to many different options. It is a powerful research tool for testing parameterization hypotheses.
The CHIMERE grid enabling process deals with the LAM management in order to deploy, boot and destroy computing nodes. The cloud enabled version of CHIMERE makes straightforward the model execution as a virtual laboratory processing unit instance: the HPC cluster, dynamically created and managed, hides the whole distributed memory message passing interface environment using pre-configured *ad hoc* virtual machine images. Moreover, the virtual laboratory offers tools for WRF/CHIMERE coupling in the data components suite.
In the past, we developed the PNAM (Parallel Naples Airsheld Model) [28], we integrated the STdEM Spatio-temporal distribution Emission Model [27] and grid enabled CAMx (Comprehensive Air quality Model with eXtension)[2].

5.3 Marine components

Currently, the ocean modeling components of our virtual laboratory primarily consist of the WW3 (WaveWatch III) [69] sea-wave propagation model. WW3 is a third generation wave model developed at NOAA/NCEP as an evolution of the WAM model [34]. It differs from its predecessors in many important points such as the governing equations, the model

structure, the numerical methods and the physical parameterizations. It is a wave-modelling framework, which allows for easy development of additional physical and numerical approaches. WW3 solves the random phase spectral action density balance equation for wave-number direction spectra. The implicit assumption of this equation is that properties of medium (water depth and current) as well as the wave field itself vary on time and space scales that are much larger than the variation scales of a single wave. Some source term options for extremely shallow water (surf zone) have been included, as well as wetting and drying of grid points. Whereas the surf-zone physics implemented so far are still fairly rudimentary the wave model can now be applied to arbitrary shallow water.
Our WW3 grid enabling process involved the development of several pre-processing and post-processing components especially related to the model coupling and the offline nesting implementation. We developed from scratch the specific components for the domain setup (CreateDomain), a sort o Geogrid equivalent for WW3, GrADS2WW3 for weather model coupling (it works with both MM5 and WRF) and the Configuration Helper component that simplifies the model setup.
Previously, for shallow water simulation, we grid-enabled the SWAN [31] in order to be coupled with WW3 and to obtain a more reliable simulation of the waves in the surf zone.
In our virtual laboratory we made available to environmental scientists two ocean dynamics models: the ROMS[63], Regional Ocean Model System, and the POM [30] (Princeton Ocean Model). The first is very well engineered, modular, suitable for an implementation on parallel distributed memory machines, and supports nesting features in order to increase the domain resolution over specified areas. POM is a simple-to-run but powerful model, able to simulate a wide-range of problems: circulation and mixing processes in rivers, estuaries, shelf and slope, lakes, semi-enclosed seas and open and global ocean. The model is a sigma coordinate, free surface ocean model with embedded turbulence and wave sub-models, and wet-dry capability. We enhanced it in a parallel version with nesting capabilities (POMpn) [44].
The POM grid enabling process required a complete revision of the code in order to implement user provided cases. In particular, we had to implement the components needed to set up spatial domains, initial and boundary conditions. An interface to a seasonal database has also been implemented to support open basins real case simulations.

5.4 Data components

Our virtual laboratory provides several processing unit components for data acquisition. The NCEPDataProvider retrieves real time weather forecast initial and boundary conditions from NOAA NCEP [14], thanks to a daemon component, completely decoupled from the grid; the ECMWFDataProvider [3] performs an on-demand download, from the ECMWF repository, of historical data for scenario and what-if analyses; the DSADataProvider performs an on-demand download of processed data. Weather stations, cams, ocean surface current radars and wind profilers are interfaced to laboratory components using the Abstract Instrument Framework and the grid enabled Instrument Service [58]. A data sink component has been developed to store and provide to users environmental multidimensional data [60].
Currently, we are working on a MET [49] grid-enabled version in order to provide a standard tool for model evaluation.
The virtual laboratory includes some data visualization tools as grid-enabled wraps on standard world spread software as GrADS [6], Vapor[18] and Ferret[4].

5.5 Operational grid application

We have used the Virtual Laboratory components to develop a grid application for producing weather and marine forecasts in both operational and on-demand mode, where several simulation models, data acquisition, conversion, and visualization software are coupled. The application workflow is simple: the starting event is produced either by an on-demand user request or by an initialization data availability signal, in an operational environment; then, the weather forecast model is initialized, and the output data is rendered by a presentation software and concurrently consumed by other models, such as ocean dynamics, sea wave propagation or air quality models; then each application branch proceeds on separate thread (Figure 4).

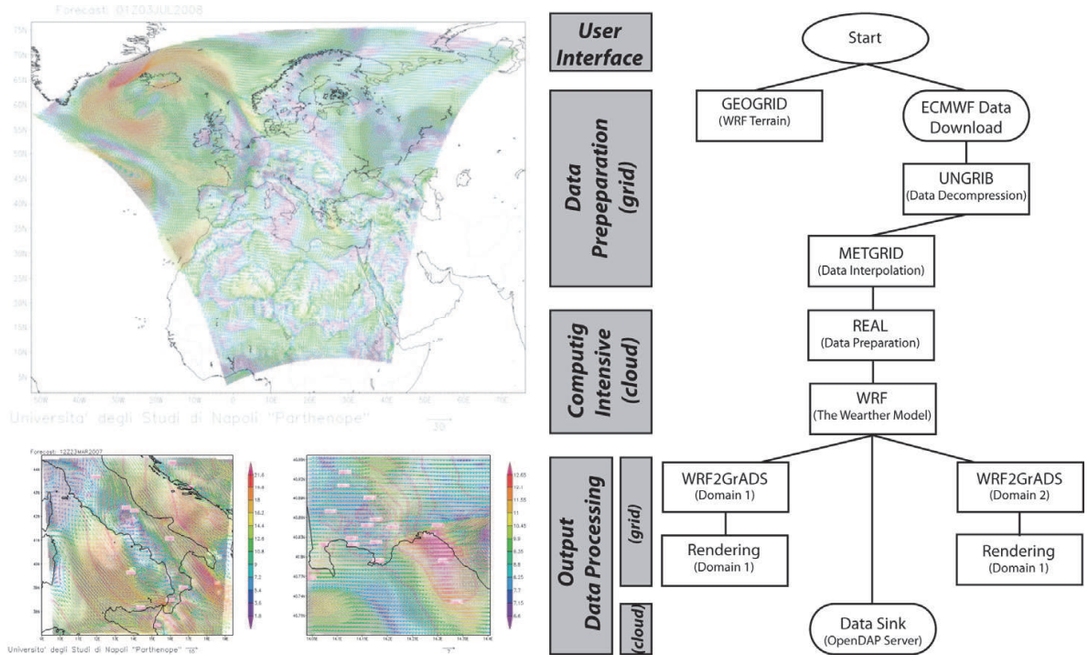


Fig. 4. a Weather Component (WRF Processing Unit) output (left) and the application workflow schema (on the right).

6. Conclusions

In this chapter we have described some of our recent results in the field of grid, cloud, and GPU computing research. We claim that our virtual laboratory for earth observation and computational environmental sciences based on hybrid grid/cloud computing technology can be a useful tool in both research and operational applications, and we have shown how to develop a complex grid application dedicated to operational weather, marine and air quality forecasts on nested spatial domains.

The virtual laboratory is an elastic-Science environment. It provides GPGPU acceleration thanks to a GPU virtualization and sharing service. This service is realized by the gVirtuS component. It enables a virtual machine instance, running in a high performance computing cloud, to properly exploit the computing power of the nVIDIA CUDA system. In discussing the architecture of our framework, the adopted design and implementation solutions, and

the key communication component, we stressed the main features of gVirtuS: the hypervisor independence, the fully transparent behaviour and, last but not the least, its overall performance. We have reported elsewhere [45] the results of an extensive test process to assess how gVirtuS performs in different and realistic setups. gVirtuS leverages on our previous works on high performance computing grids, and our interest in the apparently poor performing TCP communicator is related to other more complex deployment schemes. For example, a private cloud could be set up on a massive multi-core cluster, hosting general-purpose virtual machine instances and GPU powered computing elements for compute-intensive scientific applications.

Another important issue we have addressed in this paper is a scalable parallel I/O system for data intensive GPGPU applications. We have designed a scalable parallel I/O architecture based on a hierarchy of caches deployed on the computing virtual machines and on the I/O system backend, close to the storage. In order to hide the latency of file accesses inherent in a hierarchical architecture, we proposed a multiple level data staging strategy. The multiple level I/O pipelining in a data staging approach maps suitably to the characteristics of stream computing model, which exposes the inputs and outputs of the kernel computation, an important piece of information that can be leveraged by parallel I/O scheduling strategies.

Several scientists for different kinds of one-shot or operational applications currently use the Virtual Laboratory environment we designed and implemented. The weather forecast application we described in this chapter, deployed in its first version in 2003, runs in operational mode with a few maintenance operations, except components or grid/cloud middleware upgrades. All performed results are interactively published at a web portal http://dsa.uniparthenope.it/meteo (the actual URL could change: a redirect to elsewhere could be expected) and used by several scientists, local institutions and common users.

7. References

[1] Amazon Web Services. http://aws.amazon.com
[2] CAMx Comprehensive Air Quality Model with eXtensions. http://www.camx.com
[3] ECMWF European Centre for Medium Range Weather Forecast. http://www.ecmwf.int
[4] Eucalyptus, the open source cloud platform. http://open.eucalyptus.com
[5] Ferret data visualization and analysis. http://ferret.wrc.noaa.gov/Ferret
[6] GrADS Grid Analysis and Display System. http://www.iges.org/grads
[7] International Exascale Software Project. http://www.exascale.org.
[8] jclouds an open source framework that helps you get started in the cloud and reuse your java development skills. http://www.jclouds.org
[9] JaClouX (Java bindings for Cloud extensions) project at UniParthenope open source lab web portal. http://osl.uniparthenope.it/projects/jacloux
[10] KVM kernel based virtual machine. http://www.linux-kvm.org
[11] KVM VMChannel. http://www.linux-kvm.org/page/VMchannel_Requirements
[12] LAM/MPI local area multicomputer. http://www.lam-mpi.org
[13] libcloud a unified interface to the cloud. http://incubator.apache.org/libcloud
[14] MPICH 2 high performance and widely portable message passing interface. http://www.mcs.anl.gov/research/projects/mpich2
[15] NCEP/NOAA National Centres for Environment Prediction, National Oceanic and Atmosphere Administration. http://www.ncep.noaa.gov

[16] NVIDIA CUDA Zone. http://www.nvidia.com/cuda
[17] Top 500 list. http://www.top500.org
[18] Vapor, Visualization and Analysis Platform for Ocean, Atmospheric and Solar Research. http://www.vapor.ucar.edu
[19] VMWare. Virtualization software for desktop, servers and virtual machines for public and private cloud solutions. http://www.vmware.com
[20] VMWare. Virtual Machine Communication Interface. http://pubs.vmware.com/ vmci-sdk/VMCI_intro.html.
[21] Xen hypervisor. http://www.xen.org
[22] Ali N., P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable I/O Forwarding Framework for High-Performance Computing Systems. In Proceedings of IEEE Conference on Cluster Computing, New Orleans, LA, September 2009.
[23] Anjomshoaa A., F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, A. Savva. Job Submission Description Language (JSDL) Specification, Version 1.0. GFD-R.136. Open Grid Forum, 28-Jul-2008.
[24] Armbrust M.,A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin. I. Stoica, M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Electrical Engineering and Computer Sciences University of California at Berkeley, Technical Report No. UCB/EECS-2009-28, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html, February 10, 2009.
[25] Armbrust M.,A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin. I. Stoica, M. Zaharia. A view of cloud computing. Commun. ACM 53, 4 (Apr. 2010), 50-58, 2010.
[26] Ascione I., G. Giunta, R. Montella, P. Mariani, A. Riccio: A Grid Computing Based Virtual Laboratory for Environmental Simulations. Euro-Par 2006 Par-allel Processing, (W.E. Nagel, W.V. Nagel, W. Lehner, eds.) LNCS 4128, Springer (2006) 1085–1094.
[27] Barone G., D'Ambra P., di Serafino D., Giunta G., Montella R., Murli A., Riccio A.: An Operational Mesoscale Air Quality Model for the Campania Region. Annali Istituto Universitario Navale (2000) – 179-189, 2000.
[28] Barone G., D'Ambra P., di Serafino D., Giunta G., Murli A., Riccio A.: Parallel software for air quality simulation in Naples area. J. Environ. Manag. & Health (2000) 209–215, 2000.
[29] Bessagnet B., A.Hodzic, R.Vautard, M.Beekmann, S.Cheinet, C.Honoré, C.Liousse and L.Rouil, Aerosol modeling with CHIMERE - preliminary evaluation at the continental scale, Atmospheric Environment, 38, 2803-2817, 2004
[30] Blumberg, A.F., Mellor, G. L.: A description of a three-dimensional coastal ocean circulation model. Three-Dimensional Coastal ocean Models, edited by N. Heaps, American Geophysical Union, 1987.
[31] Booij, N., Holthuijsen, L.H., Ris, R.C.: The SWAN wave model for shallow water. Int. Conf. on Coastal Engineering., Orlando, USA (1996) 668–676, 1996.
[32] Buck I., Foley T., Horn D., Sugerman J., Fatahalian K., Houston M. and Hanrahan P. Brook for GPUs: stream computing on graphics hardware. In Proceedings of SIGGRAPH 2004.

[33] Buyya, R., Chee Shin Yeo and Venugopal, S. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. HPCC '08. 10th IEEE International Conference on High Performance Computing and Communications, 2008.

[34] Cardone V. J., J. A. Greenwood et al. The WAM model - a third generation ocean wave prediction model. J. of Phys. Oceanog., 18, 1775-1810, 1988.

[35] Dong S., G. E. Karniadakis, N. T. Karonis. Cross-Site Computations on the TeraGrid Computing in Science and Engineering, pp. 14-23, September/October, 2005.

[36] Evangelinos C. and Chris N. Hill. Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Proceeding of CCA-08, Cloud Computing and its Applications, University of Illinois in Chicago, 2008.

[37] Fagui Liu, Wei Zhou and Ming Zhou. A Xen-Based Data Sharing & Access Controlling Method. Third International Symposium on Intelligent Information Technology Application, 2009. IITA 2009. , vol.3, no., pp.7-10, 21-22 Nov. 2009.

[38] Foster I., C. Kesselman and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[39] Foster I. The Grid: A New Infrastructure for 21st Century Science. Physics Today, 55(2): 42-47, 2002.

[40] Foster I., C. Kesselman, J. Nick, S. Tuecke. Grid Services for Distributed System Integration. Computer, 35(6), 2002.

[41] Foster I. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.

[42] Foster I., Z. Yong, R. Ioan and L. Shiyong: Cloud Computing and Grid Computing 360-Degree Compared. Proceedings of Grid Computing Environments Workshop, 2008. GCE '08

[43] Giunta G., R. Montella, P. Mariani, A. Riccio. Modeling and computational issues for air/water quality problems: A grid computing approach. Il Nuovo Cimento, 28, 2005.

[44] Giunta G., P. Mariani, R. Montella, and A. Riccio. pPOM: A nested, scalable, parallel and fortran 90 implementation of the princeton ocean model. Envirnonmental Modelling and Software, 22:117–122, 2007.

[45] Giunta G., R. Montella, G. Agrillo, and G. Coviello. A gpgpu transparent virtualization component for high performance computing clouds. In P. D'Ambra, M. Guarracino, and D. Talia, editors, Euro-Par 2010 - Parallel Processing, volume 6271 of Lecture Notes in Computer Science, chapter 37, pages 379-391. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[46] Gropp, W. 2009. MPI at Exascale: Challenges for Data Structures and Algorithms. In Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing interface (Espoo, Finland, September 07 - 10, 2009). M. Ropo, J. Westerholm, and J. Dongarra, Eds. Lecture Notes In Computer Science. Springer-Verlag, Berlin, Heidelberg, 3-3, 2009.

[47] Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V., and Ranganathan, P. 2009. GViM: GPU-accelerated virtual machines. In Proceedings of the 3rd ACM Workshop on System-Level Virtualization For High Performance

Computing (Nuremburg, Germany, March 31 - 31, 2009). HPCVirt '09. ACM, New York, NY, 17-24., 2009.

[48] Hoffa C., Mehta G., Freeman T., Deelman E., Keahey K., Berriman B. and Good, J. On the Use of Cloud Computing for Scientific Workflows. eScience '08. IEEE Fourth International Conference on eScience, 2008.

[49] Holland L., J. H. Gotway, B. Brown, and R. Bullock. A Toolkit For Model Evaluation. National Center for Atmospheric Research Boulder, CO 80307

[50] Isaila F., Garcia Blas F. J., Carretero J., Wei-keng Liao, Choudhary A. A scalable MPI implementation of an ad-hoc parallel I/O system. International Journal of High Performance Computing Applications, 2010.

[51] Isaila F., Garcia Blas F. J., Carretero J., Latham R., Ross R. Design and evaluation of multiple level data staging for Blue Gene systems. In IEEE Transactions of Parallel and Distributed Systems, 2010.

[52] Karonis N., B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Journal of Parallel and Distributed Computing, 2003.

[53] Larsen, E. S. and McAllister, D. Fast matrix multiplies using graphics hardware. In Proceedings of the 2001 ACM/IEEE Conference on Supercomputing (Denver, Colorado, November 10 - 16, 2001). Supercomputing '01. ACM, New York, NY, 55-55, 2001.

[54] Lizhe Wang, Jie Tao, Kunze M., Castellanos A.C., Kramer D. and Karl W. Scientific Cloud Computing: Early Definition and Experience. HPCC '08. 10th IEEE International Conference on High Performance Computing and Communications, 2008.

[55] Mell P. and T. Grance. The NIST Definition of Cloud Computing, National Institute of Standards and Technology. Version 15. Information Technology Laboratory, July 2009.

[56] Michalakes, J., T. Canfield, R. Nanjundiah, S. Hammond, G. Grell: Parallel Implementation, Validation, and Performance of MM5. Parallel Supercomputing in Atmospheric Science. World Scientific, River Edge, NJ 07661 (1994)

[57] Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, 2004: "The Weather Reseach and Forecast Model: Software Architecture and Performance,"to appear in proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology, 25-29 October 2004, Reading U.K. Ed. George Mozdzynski, 2004.

[58] Montella R., G. Agrillo, D. Mastrangelo and M. Menna. A Globus Toolkit 4 based instrument service for environmental data acquisition and distribution. In Proceedings of the Third international Workshop on Use of P2p, Grid and Agents For the Development of Content Networks (Boston, MA, USA, June 23 - 23, 2008). UPGRADE '08. ACM, New York, NY, 21-28, 2008.

[59] Montella R., Giunta G. and Laccetti G. A Grid Computing Based Virtual Laboratory for Environmental Simulations. Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, Volume 4967/2008, 951-960, 2008.

[60] Montella R., Giunta G. and Laccetti G. Multidimensional Environmental Data Resource Brokering on Computational Grids and Scientific Clouds. Handbook of Cloud Computing, Part 4, 475-492, 2010.

[61] Montella R., Foster I. Using Hybrid Grid/Cloud Computing Technologies for Environmental Data Elastic Storage, Processing, and Provisioning. Handbook of Cloud Computing, Part 4, 595-618, 2010.

[62] Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L. and Zagorodnov D. The Eucalyptus Open-Source Cloud-Computing System. In Proceedings of the 2009 9th IEEE/ACM international Symposium on Cluster Computing and the Grid (May 18 - 21, 2009). CCGRID. IEEE Computer Society, Washington, DC, 124-131, 2009.

[63] Powell B. S., H. G. Arango, A. M. Moore, E. Di Lorenzo, R. F. Milliff and R. R. Leben. Real-time Assimilation and Prediction in the Intra-Americas Sea with the Regional Ocean Modeling System (ROMS) . Dynamics of Atmospheres and Oceans, 2009.

[64] Raj, H. and Schwan, K. 2007. High performance and scalable I/O virtualization via self-virtualized devices. In Proceedings of the 16th international Symposium on High Performance Distributed Computing (Monterey, California, USA, June 25 - 29, 2007). HPDC '07. ACM, New York, NY, 179-188, 2007

[65] Sato, T. The earth simulator: roles and impacts. Parallel Computing 30, 12 (Dec. 2004), 1279-1286, 2004.

[66] Sotomayor B., K. Keahey, I. Foster: Combining Batch Execution and Leasing Using Virtual Machines. ACM/IEEE International Symposium on High Performance Distributed Computing 2008 (HPDC 2008), Boston, MA. June 2008.

[67] Sunderam V. S. PVM: A Framework for Parallel Distributed Computing. Concurrency: Practice and Experience, 2, 4, pp 315--339, December, 1990.

[68] Thain D., T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[69] Tolman. H.L.: A third-generation model for wind waves on slowly varying, unsteady and inhomogeneous depths and currents. J. Phys. Oceanogr. , 21 (1991) 782–797, 1991.

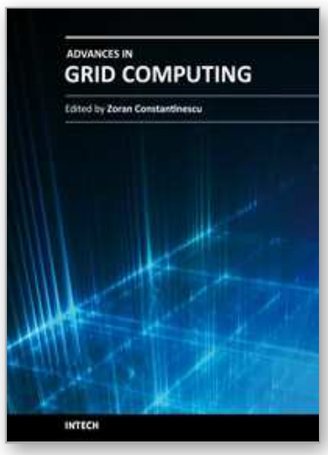
[70] Yu J. and Buyya R. A taxonomy of scientific workflow systems for grid computing. SIGMOD Rec. 34, 3 (Sep. 2005), 44-49. 2005.

[71] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. 39, 1 (Dec. 2008), 50-55, 2008.

[72] Vecchiola C., Suraj Pandey, Rajkumar Buyya. High-Performance Cloud Computing: A View of Scientific Applications. Parallel Architectures, Algorithms, and Networks, International Symposium on, pp. 4-16, 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks, 2009

[73] Wang, J., Wright, K., and Gopalan, K. XenLoop: a transparent high performance inter-vm network loopback. In Proceedings of the 17th international Symposium on High Performance Distributed Computing (Boston, MA, USA, June 23 - 27, 2008). HPDC '08. ACM, New York, NY, 109-118, 2008.

[74] Zhelong Pan, Xiaojuan Ren, Eigenmann, R. and Dongyan Xu. Executing MPI programs on virtual machines in an Internet sharing system. Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, vol., no., pp.10 pp., 25-29 April, 2006.



Advances in Grid Computing

Edited by Dr. Zoran Constantinescu

ISBN 978-953-307-301-9
Hard cover, 272 pages
Publisher InTech
Published online 28, February, 2011
Published in print edition February, 2011

This book approaches the grid computing with a perspective on the latest achievements in the field, providing an insight into the current research trends and advances, and presenting a large range of innovative research papers. The topics covered in this book include resource and data management, grid architectures and development, and grid-enabled applications. New ideas employing heuristic methods from swarm intelligence or genetic algorithm and quantum encryption are considered in order to explain two main aspects of grid computing: resource management and data management. The book addresses also some aspects of grid computing that regard architecture and development, and includes a diverse range of applications for grid computing, including possible human grid computing system, simulation of the fusion reaction, ubiquitous healthcare service provisioning and complex water systems.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen