Virtual Hybrid Elastic Clusters in the Cloud

A. Calatrava**, G. Moltó, M. Caballer, and C. De Alfonso

Instituto de Instrumentación para Imagen Molecular (I3M).
Centro mixto CSIC - Universitat Politècnica de Valéncia - CIEMAT
Camino de Vera s/n, 46022 Valencia, España
Tel. +34963877356, Fax +34963877359

Abstract. Clusters of PCs are one of the most widely used computing platforms in science and engineering, supporting different programming models. However, they suffer from lack of customizability, difficult extensibility and complex workload-balancing. To this end, this work introduces virtual hybrid elastic clusters that can simultaneously harness on-premise and public Cloud resources. They can be fully customized, dynamically and automatically enlarged and shrinked (in terms of the number of nodes) and they offer migration capabilities to outsource workload from one datacenter to another (or to a public Cloud) in different situations. This is exemplified with case studies that involve a parallel computationally intensive gyro kinetic plasma turbulence code running on such hybrid clusters with resources provisioned from an on-premise OpenNebula Cloud and the Amazon Web Services public Cloud.

Keywords Cloud Computing, Elasticity, High Throughput Computing, Cluster computing, Migration

1 Introduction

The usage of clusters of PCs as a computing facility is widespread in the scientific community with high success for both High Performance Computing (HPC) and High Throughput Computing (HTC). Nevertheless, these computing platforms suffer several drawbacks. On the one hand, physical clusters cannot be easily enlarged or shrinked, without downtime, according to the dynamic requirements of the organization. On the other hand, they lack the ability to provide a tailored execution environment customized for each application to be executed, specially when incompatible libraries are required by different applications running on the same cluster. Moreover, the ability to transparently migrate running jobs to be executed on other physical resources is not a trivial task, which is an important feature for effective workload balancing. Another important limitation is the large upfront investment together with the maintenance cost of such computing equipment for small and medium-sized research groups or organizations [1].

Traditionally, virtualization was not considered a viable option for HPC due to the overhead costs in I/O and network devices, but the major improvements in hypervisor technologies and virtualization have paved the way for Cloud computing.

** e-mail of corresponding author: amcaar@i3m.upv.es
This paradigm can solve those disadvantages with customizable virtual machines (VMs) that decouple the application execution from the underlying hardware and are dynamically provisioned and released on a pay-as-you-go basis [2]. This is the case of public Infrastructure as a Service (IaaS) Cloud providers such as Amazon Web Services (AWS) [3] or on-premise Cloud infrastructures deployed by Cloud Management Platforms (CMPs) such as OpenNebula [4], OpenStack [5] or Eucalyptus [6].

In the last years, the adoption of on-premise Cloud solutions for datacenters has increased [7], in part due to the maturity of the CMPs, which are becoming reliable enough for production-ready workloads. This enables to create virtual hybrid elastic clusters that ease extensibility and customizability in contrast to physical clusters. Virtualization also enables to partition a physical cluster into different virtual clusters specifically customized for the applications. Moreover, decoupling the application execution from the underlying hardware paves the way for migrating running applications from one hardware to another. Migration is convenient for several reasons. First of all, it enables to move workload from an overloaded datacenter (source infrastructure) to another (where the former or the latter could be a public Cloud infrastructure) to achieve workload balancing. Second, a planned maintenance on the source infrastructure might affect an application that has been running for a long time, so migration introduces the flexibility required in a modern datacenter.

This work focuses on harnessing resources from hybrid scenarios that simultaneously include resources from on-premise and public Clouds in order to create ready-to-use customized virtual elastic clusters that satisfy the dynamic computing requirements of an individual or organization. The remainder of this paper is structured as follows. First, section 2 covers the related work in this area. Next, section 3 focuses on the main characteristics of the proposed cluster deployment architecture pointing out the main scenarios where the use of this type of clusters is beneficial. Then, the paper introduces the elasticity rules employed to automatically enlarge and shrink the cluster. Afterwards, two case studies are presented in section 4 that justify the benefits of the proposed approach. Finally, section 5 summarizes the main achievements and discusses the future work.

2 Background and Related Work

There are previous works in the literature that aim at deploying virtual clusters on Cloud infrastructures. For example, StarCluster [8] enables the creation of clusters in Amazon EC2 from a predefined configuration of applications (Open Grid Scheduler, OpenMPI, NFS, etc.). Along with StarCluster a plugin called Elastic Load Balancer has been developed that is able to enlarge and shrink the cluster according to the length of the cluster’s job queue. This plugin typically runs on the local computer from which the cluster was deployed and requires permanent connection to the Cloud infrastructure, in order to create and destroy the VMs. Therefore, any disconnection on the client computer means a loss of control of the elasticity capabilities of the cluster. Instead, our proposal focuses on self-managed
hybrid clusters, where the elasticity rules are embedded in the cluster and no external entity is required. Viterraas [9] allows creating virtual clusters to manage the execution of user-defined jobs, but the user cannot remotely access the cluster. The standard distribution of Hadoop [10] includes a utility to create a virtual cluster in the Amazon EC2 infrastructure, managed by the Hadoop middleware. The utility powers on the master virtual machine, using a pre-defined Amazon Machine Image (AMI) and creates the computing nodes using another AMI, performing the required network configuration.

There are commercial solutions, like IBM Platform Dynamic Cluster [11] that aims at partitioning the owned resources to deliver each user a custom cluster with specific features. It has features such as live job migration and automated checkpoint restart. The drawback in this case is that this product is oriented to manage on-premise infrastructures and cannot be connected to commercial Cloud providers. CycleCloud [12] is a service provided by CycleComputing that deploys virtual clusters, but it is restricted to Amazon EC2. This service provides the user with a virtual cluster based on Torque [13] or Condor where a subset of popular scientific applications, offered by CycleCloud, are installed. The user is able to manage the virtual nodes using a web interface, and it is possible to configure the cluster so that it is automatically sized based upon pending jobs.

All of the above examples only consider the usage of one Cloud infrastructure (no hybrid scenarios supported). Concerning the creation of clusters over hybrid Cloud infrastructures, the authors of works such as [14], [15] and [16] have analyzed architectures, algorithms and frameworks to deploy HTC clusters over this type of infrastructures. They analyze the performance of virtual clusters deployed on top of hybrid Clouds obtaining good results that demonstrate the feasibility of these kind of deployments. The works use a fixed number of on-premise nodes, and scale out the cluster using public nodes. However, workload migration from one infrastructure to another is not considered. In [17], [18], [19], the Nimbus toolkit is employed to implement and evaluate an elastic site manager, which dynamically extends existing physical clusters based on Torque with computational resources provisioned from Amazon EC2 according to different policies. A similar approach is employed in [20], where the benefits of using Cloud computing to augment the computing capacity of a local infrastructure are investigated, but no details about the underlying technologies are given.

An important point in a hybrid environment is the interconnection between nodes that are deployed in different Clouds. The authors of [21] introduced the concept of Sky Computing, which enables the dynamic provisioning of distributed domains over several Clouds. They pointed that one of the shortcomings of this approach is the need of trusted networking environments. The same authors of the previous work [22] study the inter-Cloud communication problem across independent Clouds in detail.

Our previous work in the field is Elastic Cloud Computing Cluster (EC3) [23], a tool that creates elastic virtual clusters out of computational resources provisioned from IaaS Clouds, but no hybrid scenarios are supported. These clusters are self-managed entities that scale out to a larger number of nodes on demand, up to a maximum size specified by the user. Whenever idle resources are detected, the
clusters dynamically and automatically scale in, according to some simple policies, in order to cut down the costs in the case of using a public Cloud provider. This creates the illusion of a real cluster without requiring an investment beyond the actual usage.

3 Virtual Hybrid Elastic Clusters

The objective of the proposed architecture is to ease the deployment and management of customized virtual hybrid elastic clusters whose computational resources are simultaneously provisioned from on-premise Clouds and from different public IaaS Cloud providers. In order to have all the nodes that compose the cluster in the same subnetwork, regardless of their physical location, it is convenient to use VPN (Virtual Private Network) techniques. A VPN enables a computer on a public network, such as the Internet, to exchange data as if it was connected to the private network.

![Diagram of proposed cluster deployment architecture](image)

**Fig. 1.** Proposed cluster deployment architecture.

Figure 1 shows the proposed architecture to deploy virtual hybrid elastic clusters. The user (or the system administrator of the datacenter) requests a cluster deployment to the Infrastructure Manager (IM) [24], a component that is in charge of contacting different CMPs in order to deploy the VMs that compose the virtual cluster. As a summary, the lifecycle of the virtual hybrid elastic cluster consist of three phases: (1) the creation of the infrastructure, (2) configuring the computing nodes to behave as a cluster and (3) managing the elastic cluster.
Firstly, the user provides the IM with the initial number of nodes of the cluster, and their distribution among the on-premise and/or public Clouds accessible with his credentials. The IM is in charge of phase (1). In the scenario shown in the Figure, the frontend (head node) is deployed on the on-premise Cloud, but it could also be deployed on a public Cloud. Two corner cases are also considered, which are deploying a virtual cluster fully on a public Cloud or fully on an on-premise Cloud. The latter complies with the requirements of research communities that require instant access to cluster-based computing with data that does not leave the boundaries of the organization.

Once the computing resources for the cluster have been provisioned, the Ansible tool is employed in phase (2) for the configuration via a set of high-level recipes. This requires to configure the VPN-based network. For that, we rely on OpenVPN [25] to implement network tunnels between each individual Cloud resource and the organization’s private network. By default, the frontend of the cluster hosts a VPN server, to produce self-managed clusters that do not require external VPN services to work. However, our solution could also rely on the VPN services provided by the organization. The configuration process also installs and configures the Torque Local Resource Management System (LRMS) that is in charge of scheduling the execution of the jobs. Moreover, the dependences of the application that is going to be executed in the virtual cluster are installed and configured in this step.

When the virtual cluster is running, the users can connect via SSH to the frontend and execute their jobs. At this point starts phase (3). The workload of the cluster is periodically evaluated by the elasticity module, a component that can run inside the frontend (to produce self-managed clusters) or outside, and is in charge of monitoring the state of the LRMS queue to enforce a set of elasticity rules (described in section 3.1) to trigger actions in order to scale out (increase) or scale in (decrease) the number of nodes of the cluster. The user can choose from a set of elasticity rules as well as the maximum number of nodes of the cluster (growth limit).

From the point of view of the user, the use of a virtual hybrid elastic cluster enables proper customization of the execution environment, thus ensuring compatibility with the applications to be executed. The automatic elasticity, together with the workload balancing capability, enables to reduce the total execution time of the jobs by dynamically adapting the size of the cluster to the workload.

In addition, the system administrator is able to manage the resources of the virtual cluster with unprecedented flexibility due to the migration capabilities. For example, this enables a system administrator to temporarily outsource cluster-based workloads to a public Cloud (or to another datacenter running an on-premise Cloud) to deal with a planned outage.

3.1 Elasticity Rules

The elasticity module included in the architecture dynamically adds and removes worker nodes from the cluster by monitoring the frontend job queue. This module uses various policies to determine when to deploy additional VMs (scale out) in
the Cloud or terminate them (scale in) based on the monitored information. The behaviour of this module is described in Algorithm 1.

**Algorithm 1 Elasticity management**

**Require:** Growth limit, \( l > 0 \), scale\_out policy, scale\_in policy

```
while the frontend is running do
    Obtain the number of jobs in queue, \( j \), and the total number of nodes, \( n \)
    if \( j > 0 \) and \( n < l \) then
        Apply scale\_out policy
    end if
    if \( j == 0 \) then
        Obtain state of the nodes
        if some node state is free or offline then
            Apply scale\_in policy
        end if
    end if
end while
```

The elasticity rules or policies can be proactive or reactive. Proactive rules can be employed if job execution patterns in the clusters are known, in order to deploy and configure the nodes just in time for the execution of the jobs arriving at the LRMS. However, proactive rules typically underperform with stochastic job execution patterns. Therefore, this paper focuses mainly on reactive elasticity policies that deploy and relinquish resources based on the actual state of the cluster.

The scale out policies determine when it is necessary to increase the number of worker nodes of the cluster. Two policies are included: i) *On demand*, where for each job in the queue a worker node is deployed. Therefore, the jobs will wait for the deployment and contextualization of the node before they start their execution and ii) *Bursts*, this policy deploys a group of VMs for each job in the queue, assuming that if a job arrives at the LRMS there is an increased chance for other jobs to arrive shortly (thus including some proactivity). For HTC-based applications, such as Bag-of-Tasks or parameter sweeps, this policy is expected to reduce the average waiting time of the jobs at the expense of an increased cost (economic, in the case of public Clouds or due to idle resources in the case of on-premise Clouds).

The scale in policies determine when it is necessary to decrease the number of worker nodes. Two policies are considered: i) *On demand*, to terminate the idle worker nodes when there are no pending jobs in the LRMS and ii) *Delayed shutdown*, where idle worker nodes are terminated after a certain amount of configurable time. This is of interest when using public Clouds that bill by the hour, where idle nodes are kept available for job executions before the hour expires, even if no jobs are available to be executed at the moment.

Notice that when there are no jobs in the LRMS for a period of time, all the worker nodes will be eventually terminated, regardless of the elasticity policies, thus resulting in a cluster with only the frontend running.
4 Case Study

In order to evaluate the benefits and impact and to validate the developed system on real scenarios, two case studies are proposed using a computationally intensive parallel (hybrid MPI/OpenMP approach) scientific application. The GENE [26, 27] version 11 (release 1.7) provides a state-of-the-art nonlinear gyrokinetic solver aimed to efficiently compute the microturbulence and the resulting transport coefficients in magnetized fusion and astrophysical plasmas. The application requires an MPI-2 environment, Fortran and C compilers, and the BLAS and LAPACK libraries.

The infrastructure used to deploy the on-premise Cloud is composed by an heterogeneous blade-based system that has 4 kind of nodes: 2 x (2 quad-core L5430@2.6 Ghz, 16 GB), 2 x (2 quad-core multithreaded E5520@2.26 GHz, 16 GB), 6 x (2 quad-core multithreaded E5620@2.4 GHz, 16 GB) and 3 x (4 quad-core multithreaded E7520@1.86 GHz, 64 GB), with a total amount of 128 cores and 352 GB of RAM. The blade system is backed by a 16 TB SAN connected via a private gigabit ethernet network. This system is managed by OpenNebula 4.4, using KVM as the underlying hypervisor. With respect to the public Cloud, we relied on Amazon Web Services. The instance type chosen is m1.medium, with one (virtual) processor, 3.75 GB of RAM and 410 GB of HD. In this way, the VMs deployed on the on-premise Cloud have the same characteristics. The Virtual Machine Image (VMI) used in the on-premise Cloud provides the same execution environment as the EC2 AMI employed (ami-e50e888c). Both images provide a pristine installation of Ubuntu 12.04 LTS.

The case studies consist of jobs composed by 4 MPI processes in each node, using a maximum of 1000 MB per process. These processes communicate each other inside the node, but they do not communicate with external processes. The case studies have been downsized to get job executions in the order of minutes (instead of hours) to better focus on the dynamic cluster topology and job scheduling. The limit to the growth of the cluster has been fixed to 15 nodes.

4.1 Cloud bursting

Our first case study focuses on automatic Cloud bursting, where a cluster running on an on-premise Cloud is automatically enlarged with computational resources provisioned from a public Cloud, with no service disruption. This enables to cope with an increased workload. For that, the user deploys the cluster and submits the jobs for execution. The cluster eventually becomes overloaded and, therefore, the elasticity module in the cluster that is monitoring the status of the queue decides to deploy new worker nodes (scale out) to handle the execution of the pending tasks. The new worker nodes can be on-premise resources, which are favoured, or/and public resources if no more resources can be allocated on-premise. We are going to assume that the user is restricted to 5 VMs simultaneously running on the on-premise Cloud, and the frontend is going to be able to execute jobs (but it cannot be shutdown unless the user terminates the cluster). The goal of this case study is to analyze the behaviour of the scale in/out policies described in section 3.1.
In the three cases, the workflow is as follows: the user requests an initial cluster of 5 nodes on the on-premise Cloud to the IM. When the initial cluster is deployed and configured (minute 16-18), the user submits 10 jobs to the cluster. Regardless of the decision of the elasticity module, the user will submit two new jobs in minute 50. This behavior will be repeated in minute 85, submitting two more jobs to the cluster (14 jobs in total). It should be noticed that the average duration of each job performed in the case study is 55 minutes, regardless of the underlying infrastructure.

The bursts policy has been configured to deploy twice as much jobs were in the job queue. The behaviour of the delayed shutdown scale out policy works as follows: the on-premise nodes are terminated 10 minutes after they finished executing their jobs, if there are no more jobs waiting in the queue. Since Amazon bills by the hour, the nodes deployed in Amazon EC2 are terminated minutes before the paid hour goes by.

(a) Scale out: on demand. Scale in: on demand

(b) Scale out: on demand. Scale in: delayed shutdown

(c) Scale out: bursts. Scale in: delayed shutdown

**Fig. 2.** Evolution of cluster’s size and nodes distribution for different policies.

Figure 2 shows the behaviour of the three possible combinations of the elasticity policies. Note that the combination of the bursts (scale out) policy with on demand (scale in) policy is not possible because the extra nodes that are deployed for future jobs would be shutdown immediately by the monitor system. The elasticity rules periodically examine the job queue (every 10 seconds for these tests), executing a policy and resizing the cluster if required.
The Figure shows that in (a) there are no idle periods in the worker nodes. The lifetime of the VMs is adapted to the workload of the cluster, but every time the cluster has a job waiting in the queue, a period of deployment plus contextualization is needed (4 periods in this case, minutes 4-17, 20-40, 51-69 and 85-92). In contrast, (c) has only 2 periods of deployment and contextualization (minutes 3-15 and 20-45), but the use of the nodes is not optimized, having idle machines the most part of the time. (b) is the intermediate solution, where the utilization of computing resources is better than (c) and there are 3 periods of deployment plus contextualization (minutes 3-17, 20-43 and 51-70). The contextualization process is faster in the ONE nodes than in the EC2 nodes because of the network, since the IM is deployed in the infrastructure as the on-premise Cloud deployment.

Notice that every time a node is terminated, the cluster needs another period of contextualization in order to reconfigure and restart Torque, what requires an average of 3 minutes. A new node form the public Cloud can be included in the cluster in 15 to 20 minutes, and it does not affect to the execution time of jobs because this process is done in parallel. Similarly, when a new node is added to the deployed infrastructure, the contextualization process mainly affects to the new nodes, that need to install all the software. The rest only need to reconfigure their \texttt{/etc/hosts} file and restart Torque, but it does not affect the execution of the jobs. The total execution time of the jobs is similar in the three combinations (142 minutes in (c), 144 in (b), and 148 minutes in (a). This is because the execution of the last job burst (composed by 2 jobs) in minute 85, needs almost an hour to be executed.

### 4.2 Maintenance period in a datacenter

Maintenance periods or planned outages are very common situations in a datacenter, and they might cause several inconveniences for the users of the datacenter resources that the system administrator must deal with. Therefore, the second case study is focused on the migration capabilities of the cluster, a very useful feature from the point of view of the system administrator.

![Ordered shutdown of nodes during a planned maintenance period.](image-url)
First of all, when the user or system administrator requests a shutdown for some worker nodes, two different approaches can be used: i) Aggressive shutdown, that removes worker nodes without waiting for the completion of the jobs being executed, and ii) Ordered shutdown, which waits for the jobs to finalize their executions before shutting down the worker nodes.

An aggressive shutdown should not be used on tightly coupled parallel applications, where the loss of a worker node ruins the execution of the job unless application checkpointing is employed. However, for embarrassingly parallel independent jobs, this approach can be combined with a job resubmission system that enables to execute again the failed tasks. This, of course, affects the total execution time of the jobs. In contrast, this total execution time is not affected by using the ordered shutdown, because the system waits until the end of the jobs execution to terminate the nodes. Therefore, we will use this latter approach to illustrate the migration capabilities of our developed system.

Figure 3 shows the evolution of the distribution of nodes when the cluster is asked to migrate part of its nodes. In this case study, the chosen policies where on demand (scale out) and delayed shutdown (scale in). It started with an initial cluster of 5 on-premise nodes, where the user is executing 5 jobs (minute 18). In the minute 41, the sysadmin requests the migration of 4 worker nodes to the IM client because of a maintenance period in the physical infrastructure. So, the system disables these worker nodes, by setting their state to offline, so that they are not assigned new tasks. This state change does not affect the execution of the current jobs, which terminate in minute 67-68. One minute later, the monitor terminates the nodes, reconfigures the cluster and deploys 4 new nodes in the public Cloud. As a result, the migration process is completed in minute 73, where the 4 worker nodes that originally composed the cluster are now allocated on Amazon EC2 resources. When the user launches new jobs (minute 90), they are going to be executed by the new nodes, so the migration process was completely transparent for the user.

5 Conclusions and Future Work

This paper has introduced a software architecture that abstracts the details of cluster deployment and configuration over hybrid Clouds. The system features the provision of virtual hybrid elastic clusters, composed by on-premise resources and public resources from public Cloud providers. Moreover, the system is able to configure these resources to support the execution of the applications, and to adapt the cluster’s size and topology to the dynamic characteristics of the application and the needs of the datacenter. The benefits of the proposed architecture have been exemplified by the execution of a computationally intensive gyro kinetic plasma turbulence application, that demonstrates the feasibility of this type of architectures into de scientific community.

The future work involves improving the migration capabilities of the cluster. Different schemes are going to be considered, from the migration of the virtual clusters to another physical infrastructure (involving the migration of the frontend), to live virtual machine migration. This will introduce an unprecedented flexibility
to decouple cluster-based computations from the physical infrastructure on which the cluster was initially deployed. Moreover, this will enable datacenter migration, or the ability to migrate running computational resources from one datacenter to another. Also, we consider to develop new elastic policies that improve the performance of our virtual hybrid elastic clusters. Spot instances from Amazon EC2 are going to be considered in order to reduce the total cost over public Clouds.

Acknowledgements

This work has been developed under the support of the program "Ayudas para la contratación de personal investigador en formación de carácter predoctoral, programa VALi+d", grant number ACIF/2013/003, from the Conselleria d’Educació of the Generalitat Valenciana. Also, the authors wish to thank the financial support received from The Spanish Ministry of Economy and Competitiveness to develop the project "CLUVIEM", with reference TIN2013-44390-R.

References