EC3: Elastic Cloud Computing Cluster

Miguel Caballer[a,], Carlos de Alfonso[a], Fernando Alvarruiz[a], Germán Moltó[a]

[a]*Instituto de Instrumentación para Imagen Molecular (I3M). Centro mixto CSIC – Universitat Politècnica de València – CIEMAT, camino de Vera s/n, 46022 Valencia, España*

Abstract

This paper introduces Elastic Cloud Computing Cluster (EC3), a tool that creates elastic virtual clusters on top of Infrastructure as a Service (IaaS) Clouds. The 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 automatically scale in, according to some predefined 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. Two different case studies are presented to assess the effectiveness of an elastic virtual cluster. The results show that the usage of self-managed elastic clusters represents an important economic saving when compared both to physical clusters and to static virtual clusters deployed on an IaaS Cloud, with a reduced penalty in the elasticity management.

Keywords: Cloud computing, Service-Oriented Architectures, IaaS, Virtualization

1. Introduction

The usage of clusters of PCs as a computing facility is currently widespread in the scientific community. In the last years, the success of this computing platform, either for High Performance Computing (HPC) or for High Throughput Computing (HTC) has been unparalleled. However, one of the main drawbacks of these computing platforms is the relatively large upfront investment together with the maintenance cost. For small and medium-sized research groups or organizations the purchase of such an equipment might represent an important cost.

Traditionally, virtualization was not considered as a viable option for HPC, mainly due to the overhead costs in I/O and network devices. However, the major improvements in hypervisor technologies have paved the way for Cloud computing to rise as a paradigm where resources (in the shape of virtual machines (VM), network, storage capacity, etc.) can be dynamically provisioned and released on a pay-as-you-go basis [1]. This is the case of public Infrastructure as a Service (IaaS) Cloud providers such as Amazon Elastic Compute Cloud (EC2) or Rackspace.

*Corresponding author
Email address: micafer1@upv.es (Miguel Caballer)

In a previous work [2] we concluded that, in some cases, it is interesting to deploy a virtual cluster instead of a physical one. A virtual cluster in an IaaS provider is able to get a competitive performance per price rate, but also gets important benefits from the Cloud provider such as reducing the administration costs (both personnel costs and maintenance of equipments), avoiding hiring or buying the physical building to host the infrastructure, avoiding the upfront investments in hardware, cooling systems, etc. Therefore, deploying a cluster in the Cloud can also inherit these advantages.

However, clusters are generally not used at 100% of its capacity during their lifetime [3]. The total lifetime of a cluster can be divided into two parts: the time while the cluster is calculating (T_c) and the time when the system is idle (T_i). In a Cloud environment it would be adequate to stop the VMs while they are not being used and pay only for T_c. The idea is similar to what is currently done with energy saving techniques in a datacenter, where physical nodes are dynamically powered on and off in order to reduce energy consumption while maintaining the required level of service. However, whereas for Green computing the aim is to save energy, for Cloud computing the main aim is to save money (in the case of public Cloud providers). In this case, the Green computing techniques seem to be suitable for creating elastic clusters in an IaaS cloud deployment.

This article proposes the combination of Green computing, Cloud Computing and HPC techniques to create Elastic Cloud Computing Cluster (EC3), a tool that creates elastic virtual clusters on top of IaaS Clouds. EC3 creates elastic cluster-like infrastructures that automatically 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 cluster dynamically and automatically scales in, according to some predefined 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. Therefore, this approach aims at delivering cost-effective elastic Cluster as a Service on top of an IaaS Cloud.

The remainder of the paper is structured as follows. First, section 2 describes the related work and the main contributions of EC3 to the state-of-the-art. Later, section 3 details the architecture and implementation details of EC3. Then, section 4 describes two case studies to demonstrate the functionality of EC3 both in the case of a stable cluster and in the case of an ad-hoc cluster for a specific application. Finally, section 5 concludes the paper and points to future work.

2. Related Work

This work encompasses the dynamic deployment of virtual elastic clusters on the Cloud. Regarding the creation of clusters in the Cloud, some works such as [4], [5] and [6] have analyzed architectures, algorithms and frameworks to deploy HTC clusters over private, public and hybrid Cloud infrastructures. In [4] the authors 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 work by Wei et al [5] is focused on algorithms to deploy VMs over a set of physical resources in the most efficient way trying to obtain the best performance on the virtual cluster. In [6], the authors try obtain the best resource allocation solution for a set of virtual clusters from different users with different job features. However, none of them deals with the elastic adaptation of the cluster size to the workload submitted by their users.

In [7, 8], 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 [9], 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.

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. It is possible to add new computing nodes to the running cluster. However there is no additional support to remove the nodes from the Hadoop cluster. Finally, when the cluster is no longer needed, it is immediately terminated in order to free the allocated resources. ViteraaS [11] allows creating virtual clusters on hybrid Clouds. This software enables the submission of jobs that need to be run in a cluster to the system. The middleware creates a cluster to fit the job and manages the execution of the job. The main problem is that ViteraaS does not allow the user to remotely access the cluster. Instead, ViteraaS is devoted to execute jobs as done in classic Grid approaches without providing access to a cluster. StarCluster [12] enables the creation of a Sun Grid Engine based cluster in the Amazon EC2 infrastructure, following a predefined configuration of applications (Sun Grid Engine, OpenMPI, NFS, etc.). It includes a plugin system that enables the user to add new elements to be installed on the cluster nodes, but the VMs are based on a pre-defined Amazon AMI. As it happens with Hadoop, the creation of the cluster is made from a User Interface (UI) that connects to EC2 and starts the instances, prepares the network, configures the middleware, etc. Along with StarCluster a plugin called Elastic Load Balancer [13] has been developed that is able to grow and shrink the cluster according to the length of the cluster's job queue. The caveat of this plugin is that it requires the StarCluster UI to be connected to the Cloud infrastructure, in order to create and destroy the VMs. Thus, any failure on the UI results in the loss of control of the elasticity capabilities of the cluster.

In the case of commercial solutions we can find different approaches to create virtual clusters. One example is IBM Platform Dynamic Cluster [14] that aims at partitioning the owned resources to deliver each user a custom cluster with specific features by using virtual machines. This is different from common clusters where any application or library is installed on every node and is available for every user, and it is sometimes hard for administrators to install different applications or libraries at the same time. The virtual machines ease this situation and enable user activity isolation. The drawback in this case is that this product is oriented to manage on premise infrastructures and cannot be connected to commercial Cloud providers. Another example of commercial solutions is CycleCloud [15] that 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 SGE, Torque 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.

The summary is that several solutions have appeared for different environments, but there is no general framework that enables the creation and management of elastic clus-

ters in general IaaS deployments. Moreover (i) most of them provide a virtual cluster that comprises a fixed number of nodes; (ii) each of the solutions is oriented to a specific LRMS since they take advantage of the individual features of such implementation; and (iii) most of them are oriented to Amazon EC2 and do not consider other public IaaS deployments, or even on-premise Cloud deployments (e.g. based on OpenNebula, OpenStack, Eucalyptus, etc.).

The main advantage of our approach consists in the development of a generic framework to create and manage elastic clusters that dynamically adapt to the current workload. The deployed clusters turn into self-managed entities where elasticity is handled from the cluster itself, without requiring external entities to monitor and act upon the cluster. In this way, a cluster on a Cloud dynamically shrinks and grows according to predefined policies without human intervention. In addition, the tool seamlessly harnesses resources from public and on premise Clouds.

3. EC3: Elastic Cloud Computing Cluster

It is important to point out that the user experience should be maintained regardless of the cluster being physical or virtual, i.e., the user should be unaware that the virtual cluster is actually formed on top of a virtual infrastructure that dynamically adapts (by increasing and decreasing the number of nodes) to the cluster workload. The user is provided with a remote secure shell as the entry point to the cluster.

We have previously stated that it could be an important investment to move a cluster to a Cloud. Nevertheless, the usage of Green computing techniques in a Cloud can alleviate the cost involved in deploying and maintaining the cluster. This is achieved by introducing the concept of virtual elastic cluster, which automatically fires up and down virtual machines depending on the workload. This provides the user with the illusion of a real infrastructure at a fraction of its cost in the case of using a public IaaS Cloud provider. Virtual elastic clusters are also interesting for on-premise Cloud infrastructures as they may be used to partition a real cluster into smaller ones with specific features (Operating System, applications, libraries, etc.) that are delivered to specialized users such as scientists. In this case the elasticity is applied to multiplex the real infrastructure, and being able to show an aggregated number of nodes greater than it actually is. Such case assumes that the clusters are not being simultaneously executed using the total amount of nodes. Otherwise the risk of overcommiting will arise, but its impact is reduced if we consider that the servers are not working at 100% all the time.

Anyway it is crucial to introduce trade-off strategies that consider the orthogonal criteria of both reducing the idle nodes and reducing the waiting time of the users to the resources. The aim of this work is to combine transparency for the end users and ease of administration for the sysadmins. The sysadmin should only specify the maximum size of the cluster and delegate on automatic mechanisms to scale in and scale out the cluster depending on the current workload. As an advanced functionality, the sysadmin might indicate the precise software configuration to be available on each node of the cluster. The underlying system is responsible for deploying the infrastructure and the installation and configuration of the libraries, applications and required middlewares. The resulting system becomes autonomous and self-managed, increasing and decreasing the number of computational nodes according to the usage rates.

The following subsection describes the components employed to create the architecture required to deploy virtual elastic clusters on a Cloud.

3.1. Virtual Infrastructure Deployment

The deployment of a virtual elastic cluster broadly requires two tasks. Firstly, the virtual infrastructure must be deployed, which results in a fully configured cluster running on top of virtualized resources. Secondly, an elasticity manager must be included in the cluster in order to self-manage the elasticity rules to provision and release computational nodes on demand.

In order to deploy the virtual cluster (first task), we rely on previously developed components that help to deploy a virtual infrastructure on top of a Cloud. Even though these component are detailed in [16], we provide a concise description for each of them here for the sake of completeness.

- VMRC (Virtual Machine image Repository and Catalog). This system is used to find a suitable Virtual Machine Image (VMI) that accomplishes the requirements of the user (in terms of Operating System, CPU architecture, applications installed, etc.), and is compatible with the hypervisor available in the Cloud system. This component stores and indexes VMIs in order to be reused in multiple contexts. It also implements matchmaking algorithms to obtain a ranked list of VMIs that satisfy the aforementioned given set of requirements.

- RADL (Resource and Application Description Language). It is a declarative language for users to describe the computational infrastructure needed to run their applications. The purpose of the RADL is to describe the features that a given virtual infrastructure should have, by declaring the capabilities or requisites of the VMs to be deployed.

- Contextualizer. Ansible [17] has been used to enable the unattended execution of commands specified in an YAML document in order to perform the automated installation of software dependences. Therefore, this tool performs the installation of the software so that the VM is configured to successfully execute the application.

- Infrastructure Manager (IM): It orchestrates the different components, enabling the effective deployment of an initial computing infrastructure, and the operations required to modify it on demand, by adding or removing virtual nodes.

To provide elasticity to the virtual cluster, it is possible to modify the source code of an LRMS that considers energy management to support shutting down the nodes, in order to power on or off virtual machines instead of real nodes. An alternative is to take advantage of existing energy management software to create ad-hoc solutions for specific combinations of LRMS and Cloud deployments. Some examples of such LRMS are MOAB (the Enterprise version of Maui) [18] or SLURM [19]. In this work, the Cluster Energy Saving system (CLUES) [20], [21] has been used. CLUES is an energy management system for High Performance Computing (HPC) Clusters and Cloud infrastructures. The main function of the system is to power off internal cluster nodes when they are not being used, and conversely to power them on when they are needed. CLUES is integrated with the cluster management middleware, such as a LRMS or a

Cloud infrastructure management system, by means of different connectors. CLUES is also integrated with the physical infrastructure by means of different plug-ins, so that nodes can be powered on/off using the techniques which best suit each particular infrastructure. It also provides a hooking system enabling actions to be executed before or after an action is done by the CLUES scheduler.

Section 3.3 will describe the whole process of creating the infrastructure by orchestrating all these components.

3.2. Elasticity Rules

Depending on the usage patterns, each cluster might rely on different elasticity rules (e.g. having available as many nodes as required to match the number of jobs to be executed, having a specified queue size, etc.). Therefore, it is important to provide different policies in order to match the requirements of the sysadmins.

These policies are related to the elasticity manager which, in our case is handled by CLUES. This software implements different policies that aim at balancing the tradeoff that arises when trying to minimize the waiting time for the jobs (which involves a larger number of available nodes) and the minimization of a Cloud infrastructure cost (which involves a reduced number of nodes, which generate a cost).

In our case, we have modified the CLUES scheduler to adapt it to the purpose of the virtual elastic cluster and to introduce new policies. The policies can be divided in two groups: the policies used to decide when to increase the capacity of the cluster and those used to decide when to decrease the number of nodes.

In the first case CLUES can interact with the LRMS at two levels. On the one hand, it intercepts the submitted jobs before they reach the LRMS. In this way, CLUES can decide if it is required to increase the capacity of the cluster to make room for the job. When the submitted job reaches the LRMS, the cluster will have the appropriate size to accommodate the workload (depending on the elasticity rule, the job might end up on the queue waiting for other jobs to finish). On the other hand, CLUES also monitors the queued jobs in the LRMS to check if these jobs need new nodes to be added to the cluster.

The next paragraphs describe the different elasticity rules for the intercepted jobs and for the jobs queued up at the LRMS. It also addresses some considerations in order to shut down the virtual machines.

3.2.1. Starting nodes for intercepted jobs

For the intercepted jobs, these policies provide them with the illusion of a cluster with the required capacity, since nodes will be automatically started (i.e., virtual machines will be automatically deployed and included in the cluster as internal nodes) before the job is scheduled to run on these nodes via the LRMS. By intercepting jobs, it is possible to give an immediate response to the user, before the job arrives to the queue system. In contrast with other solutions that periodically polls the information about the LRMS queues that have some kind of delay on detecting the new jobs.

These are the policies that have been implemented in EC3 by means of integrating the CLUES software, for intercepted jobs:

- Group-based start. Every time a new internal node is required, a group of them are started. This policy assumes a workload model in which as soon as a job reaches

the LRMS, there is a high probability that other subsequent jobs will be submitted in a short period of time. By overprovisioning a larger number of nodes, the waiting time of the subsequent jobs will be reduced. This represents an appropriate policy when a sudden burst of relatively large jobs are submitted to the cluster. By properly adjusting the size of the group, the waiting time of the jobs, together with the related cost, might be reduced.

- Starting the whole cluster. When a job is submitted, the whole cluster is started. This is useful in the cases where the configuration of the cluster requires all the nodes to be active in order to properly operate This policy also pays off when workload peaks occur, since the whole cluster might cope with the workload if it has been properly dimensioned. It is important to point out that idle nodes will be later shut down according to the inactivity rules of the elasticity manager.

Notice that it is difficult to choose a one-size-fits-all policy for a general cluster. However, virtual clusters on a Cloud can be deployed to accommodate specific workloads that arise in scientific applications, specially for High Throughput Computing (HTC) schemes such as parameter sweep or Bag of Tasks (BoT), and High Performance Computing (HPC).

3.2.2. Starting nodes for jobs in the LRMS

For the jobs that are already queued up at the LRMS, the following policies to start nodes are oriented to maximize the energy saving (in the case of a physical cluster) or the cost (in the case of a virtual cluster in a Cloud):

- When a job has remained in the queue for a certain amount of time (X). This policy tries to prevent jobs from exceeding a threshold waiting time, increasing the size of the cluster when this rule is triggered. In this case, the requirements of the job are reevaluated (as if it was submitted again) to decide if new nodes should be provisioned in order to satisfy them.

- When the queue size is larger than N jobs during X time units. This policy aims at maximizing the energy saving (or the cost), since new nodes are only provisioned when the cluster workload is relatively large for a sustained amount of time. This represents a tradeoff between the quality of service in the cluster (in terms of reduced waiting time for the jobs) and the economic savings. Therefore, there is a limit in the number of jobs waiting to be executed, but a large number of internal computing nodes is avoided.

3.2.3. Shutting down nodes that are not in use

Deciding when to shut down a node is a difficult task because it has an impact on the end user, since starting the node again forces to wait an extra time for subsequent jobs. Concerning the policies used to decide when to decrease the number of nodes, the strategy is to remove a node from the cluster when it has been idle for a specified amount of time. Increasing this time results in an increased cost of the cluster, although the waiting time of the jobs will probably decrease. The selection of this time depends on the workload of the cluster and it is important to achieve a good tradeoff between the cost and the waiting time of the jobs. In any case it is important to consider the following aspects:

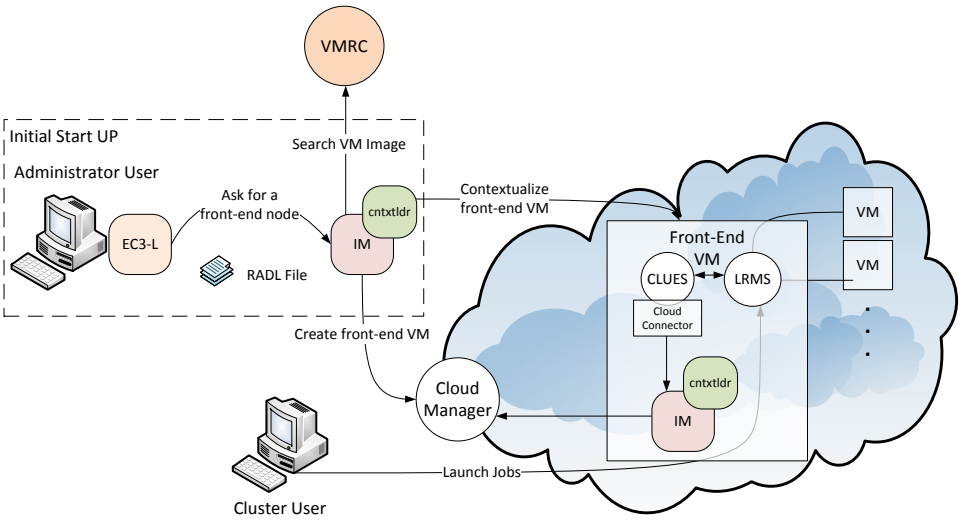


Figure 1: EC3 architecture.

- Time blocks. In some cases, such as when using public Clouds, the user pays a (typically) hourly-rate for using VMs. Therefore, once this block of time has been payed it is a good option to maintain the VM active until the whole block of time has expired (i.e. hour or fraction). Considering time blocks enables the system to keep the maximum number of nodes active in order to try to minimize the waiting time of jobs without an additional cost.

- Queued jobs. The queued jobs at the LRMS must be considered before removing any node from the cluster. If a number of idle nodes are going to be necessary for the execution of queued jobs, those nodes are not shut down. This strategy avoids an increased waiting time for queued jobs when the scheduler decides to run them. However, it may increase the cost due to the number of idle nodes not removed.

- Keeping some nodes always active. This strategy enables the cluster to have at least n idle nodes waiting for jobs. This way, we try to prevent incoming jobs from waiting while internal computing nodes are started. This is a good policy if the cluster workload consists of series of relatively small jobs. The value of n may also depend on the time needed to boot up new nodes. If the time needed is small, a smaller value of n could be used. In this way, there is a tradeoff between the cost of the initially idle nodes and the ability to accommodate jobs with minimum waiting time.

3.3. Overall Architecture

Figure 1 summarizes the main architecture of EC3. As stated earlier, the deployment of the virtual elastic cluster consists of two phases. The first one involves starting a VM

in the Cloud to act as the cluster front-end while the second one involves the automatic management of the cluster size, depending on the workload and the specified policies.

For the first step, a launcher (Elastic Cloud Computing Cluster Launcher or EC3-L) has been developed that deploys the front-end on the Cloud using the infrastructure deployment services described in section 3.1. The sysadmin will run this tool, providing it with the following information:

- Maximum cluster size. This serves to establish a cost limit in case of a workload peak. The maximum cluster size can be modified at any time once the virtual cluster is operating. Thus, the sysadmins can adapt the maximum cluster size to the dynamic requirements of their users. In this case the LRMS must be reconfigured to add the new set of virtual nodes and in some cases it may imply a LRMS service restart.

- RADL document specifying the desired features of the cluster front-end, regarding both hardware and software (OS, LRMS, additional libraries, etc.). These requirements are taken by the launcher and extended to include additional ones (such as installing CLUES and its requirements together with the libraries employed to interact with the IaaS Cloud provider, etc.) in order to manage elasticity.

The launcher starts an IM that becomes responsible of deploying the cluster front-end. This is done by means of the following steps: (i) selecting the VMI for the front-end. The IM can take a particular user-specified VMI, or it can contact the VMRC to choose the most appropriate VMI available, considering the requirements specified in the RADL; (ii) choosing the Cloud deployment according to the specification of the user (if there are different providers); (iii) submitting an instance of the corresponding VMI and, once it is available, installing and configuring all the required software that is not already preinstalled in the VM. One of the main LRMS configuration steps is to set up the names of the cluster nodes. This is done using a sysadmin-specified name pattern (e.g. vnode-*) so that the LRMS considers a set of nodes such as vnode-1, vnode-2... vnode-n, where n is the maximum cluster size. This procedure results in a fully operational elastic cluster.

Once the front-end and the elasticity manager (CLUES) have been deployed, the virtual cluster becomes totally autonomous and every user will be able to submit jobs to the LRMS, either from the cluster front-end or from an external node that provides job submission capabilities. The user will have the perception of a cluster with the number of nodes specified as maximum size. CLUES will monitor the working nodes and intercept the job submissions before they arrive to the LRMS, enabling the system to dynamically manage the cluster size transparently to the LRMS and the user, scaling in and out on demand. The scale functionality is provided by CLUES with a developed connector that interacts with different IaaS Clouds, as will be explained in section 3.4.

Just like in the deployment of the front-end, CLUES internally uses an IM to submit the VMs that will be used as working nodes for the cluster. For that, it uses a RADL document defined by the sysadmin, where the features of the working nodes are specified. Once these nodes are available, they are automatically integrated in the cluster as new available nodes for the LRMS. Thus, the process to deploy the working nodes is similar to the one employed to deploy the front-end.

Note that the EC3-L tool can be executed on any machine that has a connection with the Cloud system and it is only employed to bootstrap the cluster. Once deployed, the cluster becomes autonomous and self-managed, and the machine from which the EC3-L tool was used (the dashed rectangle in Figure 1) is no longer required. The expansion of the cluster while it is operating is carried out by the front-end node, by means of CLUES, as explained above.

On the other hand, even though the front-end duties could be assumed by a machine outside of the Cloud, deploying the front-end together with the working nodes in the Cloud has several advantages: (i) there is direct connectivity between the front-end and the working nodes, without requiring Virtual Private Network (VPN) tunnels. In addition, the connectivity among the nodes does not generate a cost, since traffic does not crosses the boundaries of the public Cloud provider (this is the case of a region in Amazon EC2, for example) and it increases communication speed between the front-end and the working nodes; (ii) it avoids managing the physical machine together with its drawbacks (power outages, network problems, hardware problems, etc.), since these issues are delegated (never avoided) to the Cloud provider; (iii) this will conform a self-managed cluster on the Cloud, independently from other external machines.

3.4. Connecting to the IaaS

In order to manage the elastic cluster on top of a Cloud infrastructure, we created the appropriate CLUES Cloud connector which enables the interaction with virtual machines instead of physical ones. This connector interacts directly with the IM for the deployment of VMs in the Cloud and, therefore, it provides a gateway to operate with the different Cloud middlewares supported by the IM (currently supporting Amazon EC2, OpenStack and OpenNebula) and standards (OCCI), contacting the corresponding Cloud Manager. The user must provide the appropriate credentials to access the different Cloud deployments. The connector will pass these credentials to the IM, enabling access to the Cloud providers.

The Cloud connector is in charge of starting and stopping the machines requested by CLUES. In this case, instead of dealing with physical machines, it deploys or terminates instances of virtual machines in the Cloud infrastructure. Besides, an additional step is required to integrate new VMs into the LRMS. In particular, the LRMS software must be installed and configured in the newly created VM, and the front-end must be reconfigured to include the new node. This last step includes mapping the IP of the new Cloud instance to one of the configured virtual nodes that are initialized with an unreachable IP, to enable the front-end node to access it. All these steps are defined via a recipe that is processed by the contextualizer software, and they are configurable so that the users can adapt it to their own configuration requirements. In case of terminating an instance, no reconfiguration is needed since the LRMS will detect that the node is down, and it will be discarded to receive new jobs.

In addition to the aforementioned tools to start and terminate nodes, the Cloud plugin includes a monitoring system that checks the state of the submitted VMs, together with the monitoring of the LRMS available in CLUES. This is employed to prevent having active VMs (generating cost) that are not integrated in the LRMS (which have been disconnected from the system). In that case, these VMs can be terminated if the LRMS reports that they are in a wrong state for a certain amount of time, defined by the sysadmin.

4. Case studies

In order to assess the effectiveness of an elastic virtual cluster on a Cloud infrastructure we have used two different use cases. The first one tries to analyze the usage of the EC3 solution in the case of two clusters during a long usage period. It uses the analysis made in [21] with two physical clusters and applies the same analysis to a "Cloud version" of the clusters. The second one uses EC3 to create an ad-hoc cluster to execute an HTC-based scientific application with dynamic computing requirements, which can greatly benefit from an elastic computing infrastructure that adapts to the workload changes as the execution progresses

4.1. Clusters with long usage period

In [21], CLUES was tested with two real use cases, involving two physical HPC clusters of 51 and 7 nodes. The first one (Cluster 1) is composed of 51 bi-processor nodes with Intel Xeon CPUs at 2.80GHz, interconnected by a SCI network in a 10x5 2D torus topology. Each node has 2 GB of RAM memory. The second one (Cluster 2) is composed of an M1000e blade server chassis with 7 Dell M610. Each M610 node has two quad-core Intel Xeon E5620 processors, making a total of 8 cores and 16 GB of RAM per node. It was shown that significant energy/cost savings could be obtained by using green computing techniques (38% saving in the first cluster and 16% in the second one). In this section the idea is to explore what is the cost of running the same clusters on Amazon EC2 and, especially, what is the saving obtained by using the elasticity features provided by EC3/CLUES. The instance types *c1.medium* and *m3.2xlarge* respectively have been selected since they are the most similar ones to the original physical nodes. For simplicity, the use case does not consider using reserved instances to decrease the cost of the allocated resources in the Cloud. However, as shown in [2], a proper selection of the number of reserved instances, considering the cluster workload, can deliver significant cost savings.

In [21], the power consumption of the clusters was measured to get the energy consumed and its associated cost. In particular, three different states for a cluster node were considered: switched off ("N. off"), switched on but idle ("N. idle"), and fully used ("N. used"). The clusters worked using CLUES for a period of eight months, and the percentage of time a node had stayed on average in each state was obtained. Then, the total cost of the energy consumed was derived.

In order to obtain the cost of the same clusters if they are deployed on EC2, we consider the cost of a virtual node instance running for a period of one year. This cost is shown in the rightmost column of Tables 1 and 2. Additionally, since the cluster uses the elasticity features of EC3, we need to know the percentage of time a node stays in each state. These percentages can be expected to be the same as in [21], if we assume that the computational power and boot time of a virtual node are approximately the same as those of a physical node. Some studies such as [22] show that the mean boot time of an EC2 instance is about 60-90 seconds, which is consistent with the boot time of the clusters considered in [21].

In accordance to the previous discussion, tables 1 and 2 show the cost of the two clusters if they are deployed on EC2 for a period of one year. The left part of the table contains the data for the case of an elastic cluster (EC3) and the center part corresponds to the case of a static cluster. In each of these two cases, the column "PCT" represents

	Elastic cluster (EC3)		**Static cluster**		**Cost per node**
	PCT	**$**	**PCT**	**$**	**$**
N. Off	52.0%	0	0.0%	0	0
N. Idle	12.7%	8,066	64.7%	41,091	1,270
N. Used	35.3%	22,419	35.3%	22,419	1,270
Other	100.0%	1,270	100.0%	1,270	1,270
TOTAL		31,755		64,780	

Table 1: Cluster 1 cost

	Elastic cluster (EC3)		**Static cluster**		**Cost per node**
	PCT	**$**	**PCT**	**$**	**$**
N. Off	55.6%	0	0.0%	0	0
N. Idle	11.6%	6,097	67.2%	35,320	8,760
N. Used	32.8%	17,240	32.8%	17,240	8,760
Other	100.0%	8,760	100.0%	8,760	8,760
TOTAL		32,097		61,320	

Table 2: Cluster 2 cost

the percentage of time a node spent on average in each state (taken from [21], as discussed above), and the column "$" contains the amount of money dedicated to keep the node in that state. Note that in the case of a static cluster a node is never shut down, so the percentage of time in that state is added to that of the idle state. The row "Other" refers to the cost of the essential components of the cluster that are always on, in this case the front-end instance. The results show that the total amount of money saved is 33,025 $ in cluster 1 and 29,223 $ in cluster 2, which represents 49% and 52% of the total amount of money, respectively.

On the user impact side, the results shown in [21] are also applicable here, which means that only 1.31% and 2.9% of total jobs of clusters 1 and 2, respectively, needed to wait to access the resources. The average waiting time for these jobs was 1'40' and 1'54' respectively.

4.2. Ad-Hoc Cluster

For this particular case study, we have used a scientific application that performs the optimization of photonic crystal fibres using automated procedures based on Genetic Algorithms [23]. The large amount of computations involved during the optimization process demand the usage of efficient technologies that are able to cope with these computational requirements. Genetic Algorithms (GA) are employed to optimize the features of the fiber and the corresponding injected pulse. It is well known that increasing the number of individuals increases the search capabilities of the GA, although this increases the computational cost in order to evaluate the fitness of all individuals [24]. Therefore, a scheme where the population size of the GA shrinks and grows according to the evolution of the fitness function is of importance in order to tradeoff computational cost

and enhanced search capabilities. For this case study, this generates several jobs to be executed where each job corresponds to the evaluation of a single individual.

Two different cloud deployments have been used to perform this second test. The first one is a private Cloud managed by OpenNebula 3.4.1 using the KVM hypervisor. Amazon EC2 has been selected as the second cloud deployment. The test includes a total of 52 job executions in 8 series of 2, 4, 8, 10, 12, 6, 8 and 2. This execution pattern simulates the changes in a dynamic population of a GA of up to 12 individuals in which the population increases or decreases in each generation. The case study has been slimmed down in order to focus on the execution patterns rather than addressing a large problem. With a larger amount of computational resources, a larger population size can be employed. In this particular application, a population size of 30-50 is typically employed.

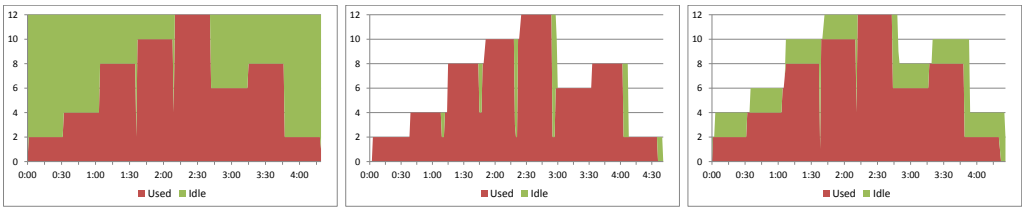
Six different tests have been carried out. In the first two (cases a and d in Figure 2), the virtual cluster is kept fully operational during the whole duration of the test. In the other four (cases (b), (c), (e) and (f) in Figure 2), CLUES is employed to dynamically manage the cluster size depending on the workload, using two different policies to increase the size of the cluster. In particular, cases (b) and (e) use a group-based start policy of size 2, while cases (c) and (f) use a policy of keeping 2 extra nodes alive. Concerning the termination policy of these four tests, a reduced waiting time was employed (5 min.), in order to minimize the number of VMs up during all the execution time, reducing the usage of the Cloud platform. In the case of the EC2 tests, as the price is charged per instance-hour, the EC3 platform does not terminate the instances until the whole hour has passed.

In the OpenNebula (ONE) tests, Figure 2 shows the number of existing VMs in Used and Idle states during the execution time. In the EC2 tests there is also a third state named "Paid". This state corresponds to idle VMs that are not terminated because their paid hour is not over yet (with a margin of 5 min.).

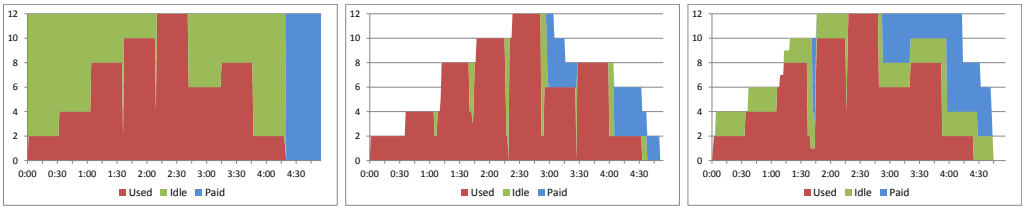
In the case (a), the global execution time reaches a total of 4h19', with an average job time of 31'27". Since a total of 12 nodes remain active during the whole execution process, this represents a total 51h48' of CPU time, where 46.31% of the time is effective and the remainder 53.69% corresponds to idle time for the VMs. In the EC2 case (d), the total CPU time consumed increases to 59h due to the instance-hour price. 40.68% of the total time is effective, 47.16% corresponds to idle VMs, and 12.16% is for VMs that are kept idle until the end of the paid hour.

In the case (b) (using OpenNebula and CLUES with a group-based start policy of size 2), the total execution time rises up to 4h36', due to the waiting time of some jobs while the VMs are started and get ready to accept the execution of jobs. This represents 6.5% extra time. In the tests performed, a total of 14 jobs have to wait for a node to start, with an average waiting time of 2'47". During the execution, there are six moments that require nodes to be started, which provokes that extra time. In this case, the effective usage rate of the cluster is 50.9% but the main difference with respect to the previous case is that the nodes are idle for only 3.4% of the time, since CLUES is responsible to terminate the nodes for the remainder 45.7%, thus without generating an excessive usage. Therefore, a total 29h51' of CPU time is required from the 12 virtual nodes, which represents a saving of 42.3% of CPU time.

In the case (e) (the same as (b) but EC2 instead of ONE), the total execution time is almost the same (4h35'), but in this case only 12 jobs have to wait for a VM to start,



(a) ONE. Static size, 12 nodes (b) ONE. Dynamic size, group-based start policy (size 2) (c) ONE. Dynamic size, keeping at least 2 nodes alive



(d) EC2. Static size, 12 nodes (e) EC2. Dynamic size, group-based start policy (size 2) (f) EC2. Dynamic size, keeping at least 2 nodes alive

Figure 2: Node usage with different configurations of the virtual clusters with up to N nodes.

with an average waiting time of 2'56". This is due to the fact that some VMs that are kept idle until the end of the paid hour are reused for other jobs (this happens in the transition from 6 to 8 nodes at approximately 3h30'). As in the previous EC2 test, the total CPU time used increases to 34h27', compared with the OpenNebula test, due to the instance-hour price. There is a saving of 47.37% of money with respect to the static EC2 test (d).

In the case (c) (using OpenNebula and CLUES with 2 extra nodes alive), the total execution time is 4h26', that represents 2.7% extra time with respect to the case (a). This is clearly lower than the case (b), due to the use of the policy to keep 2 extra nodes alive, reducing to two the number of moments that require nodes to be started. In this case, the main difference with respect to the case (b) is that 16.1% of the time the nodes are idle, and 34.9% are off. A total of 34h36' of CPU time is required from the 12 virtual nodes, which represents a saving of a 33.2% with respect to having the whole cluster on.

Finally in the case (f) (the same as (c) but EC2 instead of ONE), the total execution time is very similar to (c) with 4h28'. Like in the other EC2 tests, the total CPU time increases, in this case to 44h23', which still represents a saving of 28.2% of money compared to the static EC2 test (d).

As expected, using the elastic cluster considerably reduces the usage of the private Cloud infrastructure and the total cost of the execution on a public IaaS Cloud provider, although a small increase in the execution time is introduced. It depends on the selected policy to obtain a lower extra time or a lower cost. The first policy (cases (b) and (e)) obtains a greater cost reduction but it introduces some extra execution time, because this strategy only adds the required nodes on demand. In the second one (cases (c) and (f)) the cost reduction is less important but the extra execution time is reduced, because keeping 2 extra nodes alive (with this particular workload) enables a reduction in the number of times the jobs must wait. The impact in the execution time between the two

different policies is very low (below 4% of total time).

Another important issue to choose the right policy is to forecast the time needed to add a node into the cluster. In the previous tests, the time needed to have a node running and configured into the LRMS is 2'47" on average in the OpenNebula case and 2'56" in the EC2 tests. But it is important to notice that this time may have important variations. In case of using a large infrastructure such as Amazon EC2, some studies [22] have demonstrated that launching simultaneous VMs has little impact in the time required to have the VMs up and running. However, in private infrastructures some factors may affect this time such as the size of the Cloud deployment, the number of running VMs, the usage of the network, the policy used to balance the workload, etc. For example, launching a set of VMs to start simultaneously has an important impact in this time. Therefore, if the block size selected to add nodes to the cluster is large, and the Cloud infrastructure is relatively small, then the time may have an important increase.

5. Conclusion and Future work

This article has introduced Elastic Cloud Computing Cluster (EC3), a tool to create HPC clusters on top of Cloud infrastructures that, using Green computing concepts, creates a self-managed system that dynamically scales to adapt to the workload of users.

The tool builds on top of CLUES, an energy manager system for cluster-like infrastructures, enabling the cluster to gain elastic capabilities on a Cloud deployment. For that, a Cloud connector has been developed to manage the interaction with the infrastructure manager in charge of starting and terminating the virtual machines that correspond to the internal nodes. Moreover the CLUES scheduler has been modified to adapt its policies to be able to manage elasticity to create virtual elastic clusters.

EC3 enables end users to deploy elastic clusters of a given maximum size in a matter of minutes with just a command line tool. Besides, the policies to start and shutdown nodes have been contributed back in order to enhance CLUES enabling more possibilities in order to consider the tradeoff between minimizing the impact in the waiting time for the user (which implies having a larger number of nodes started) and the reduction of the Cloud costs (which implies having a reduced number of running virtual machines).

Future works include several research lines. In the case of commercial Clouds it is crucial to incorporate cost-aware schedulers that consider not only the number of nodes but also the cost of the instances in order to better manage the budget without surpassing the specified limits (daily, monthly, etc.) specified by the user. Concerning the costs, it is important to consider deploying multi-core virtual machines that can share the execution of several jobs. This would enable reduced prices per execution unit. Finally, it is important to consider the case for heterogeneous cluster computing, where each node could exhibit different capabilities (more storage space, different CPU type, support for GPUs, etc.).

Acknowledgment

The authors would like to thank the financial support received from the Generalitat Valenciana for the project GV/2012/076 and to the Ministerio de Ciencia e Innovación for the project CodeCloud (TIN2010-17804)

References

[1] P. Mell, T. Grance, The NIST Definition of Cloud Computing. NIST Special Publication 800-145 (Final), Tech. rep. (2011).
URL http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[2] C. de Alfonso, M. Caballer, F. Alvarruiz, G. Moltó, An economic and energy-aware analysis of the viability of outsourcing cluster computing to a cloud, Future Generation Computer Systems 29 (3) (2013) 704 – 712. doi:10.1016/j.future.2012.08.014.
URL http://www.sciencedirect.com/science/article/pii/S0167739X12001720

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, Above the clouds: A berkeley view of cloud computing, Tech. rep., UC Berkeley Reliable Adaptive Distributed Systems Laboratory (2009).
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.7163\&rep=rep1\&type=pdf

[4] R. S. Montero, R. Moreno-Vozmediano, I. M. Llorente, An elasticity model for High Throughput Computing clusters, Journal of Parallel and Distributed Computing 71 (6) (2011) 750–757. doi:10.1016/j.jpdc.2010.05.005.
URL http://dl.acm.org/citation.cfm?id=1975006.1975198

[5] X. Wei, H. Wang, H. Li, L. Zou, Dynamic Deployment and Management of Elastic Virtual Clusters, in: 2011 Sixth Annual Chinagrid Conference, IEEE, 2011, pp. 35–41. doi:10.1109/ChinaGrid.2011.31.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true\&arnumber=6051755\&contentType=Conference+Publications

[6] F. Liu, X. Dong, A Novel Elastic Resource Allocation Strategy of Virtual Cluster, in: 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming, IEEE, 2011, pp. 168–173. doi:10.1109/PAAP.2011.28.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true\&arnumber=6128497\&contentType=Conference+Publications

[7] P. Marshall, K. Keahey, T. Freeman, Elastic site: Using clouds to elastically extend site resources, in: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2010, pp. 43–52.

[8] P. Marshall, H. Tufo, K. Keahey, D. Labissoniere, M. Woitaszek, Architecting a Large-Scale Elastic Environment: Recontextualization and Adaptive Cloud Services for Scientific Computing, in: 7th International Conference on Software Paradigm Trends (ICSOFT), Rome, Italy, 2012.

[9] M. De Assuncao, A. Di Costanzo, R. Buyya, Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, in: Proceedings of the 18th ACM international symposium on High performance distributed computing, ACM, Garching, Germany, 2009, pp. 141–150.
URL http://portal.acm.org/citation.cfm?id=1551635

[10] A. Bialecki, M. Cafarella, D. Cutting, O. O'Malley, Hadoop: a framework for running applications on large clusters built of commodity hardware, Tech. rep. (2005).
URL http://hadoop.apache.org

[11] F. Doelitzscher, M. Held, C. Reich, ViteraaS: Virtual Cluster as a Service, Technology and Science.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6133210

[12] MIT, StarCluster.
URL http://web.mit.edu/stardev/cluster/

[13] MIT, StarCluster Elastic Load Balancer.
URL http://web.mit.edu/stardev/cluster/docs/0.92rc2/manual/load_balancer.html

[14] IBM, IBM Platform Dynamic Cluster.
URL http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/dynamiccluster.html

[15] CycleComputing, CycleCloud.
URL http://www.cyclecomputing.com/cyclecloud

[16] C. de Alfonso, M. Caballer, F. Alvarruiz, G. Moltó, V. Hernández, Infrastructure Deployment Over the Cloud, in: 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011), 2011, pp. 517–521.

[17] AnsibleWorks, Ansible.
URL http://ansible.cc

[18] Cluster Resources Inc. Green Computing Powered by Moab.
URL http://www.clusterresources.com/solutions/green-computing.php

[19] Simple Linux Utility for Resource Management. Power Saving Guide.
URL https://computing.llnl.gov/linux/slurm/power_save.html

[20] C. de Alfonso, M. Caballer, V. Hernández, Efficient power management in high performance computer clusters, in: 1st International Multi-Conference on Innovative Developments in ICT, 2010, pp. 39–44.

[21] F. Alvarruiz, C. de Alfonso, M. Caballer, V. Hernandez, An energy manager for high performance computer clusters, in: Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on, 2012, pp. 231 –238.

[22] S. Ostermann, A. Iosup, N. Yigitbasi, A performance analysis of EC2 cloud computing services for scientific computing, in: G. Avresky, Dimiter R. and Diaz, Michel and Bode, Arndt and Ciciani, Bruno and Dekel, Eliezer and Akan, Ozgur and Bellavista, Paolo and Cao, Jiannong and Dressler, Falko and Ferrari, Domenico and Gerla, Mario and Kobayashi, Hisashi and Palazzo, Sergio and Sa (Ed.), Cloud Computing, Springer Berlin Heidelberg, 2010, pp. 115–131.
URL http://www.springerlink.com/index/T640753R2597524U.pdf

[23] G. Moltó, M. Arevalillo-Herráez, C. Milián, M. Zacarés, V. Hernández, A. Ferrando, Optimization of Supercontinuum Spectrum Using Genetic Algorithms on Service-Oriented Grids, in: Proceedings of the 3rd Iberian Grid Infrastructure Conference (IberGrid 2009), 2009, pp. 137–147.

[24] J. Arabas, Z. Michalewicz, J. Mulawka, GAVaPS-a genetic algorithm with varying population size, in: Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, IEEE, pp. 73–78. doi:10.1109/ICEC.1994.350039.
URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=350039