Self-managed Cost-efficient Virtual Elastic Clusters on Hybrid Cloud Infrastructures[☆]

Amanda Calatrava[a,*], Eloy Romero[a], Germán Moltó[a], Miguel Caballer[a], Jose Miguel Alonso[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
Tel. +34963877356, Fax +34963877359

Abstract

This paper describes the developments to produce EC3 (Elastic Cloud Computing Cluster), a tool that creates self-managed cost-efficient virtual hybrid elastic clusters on top of Infrastructure as a Service (IaaS) Clouds. Using spot instances, together with checkpointing techniques, EC3 can significantly reduce the total cost of executions while introducing automatic fault tolerance. Moreover, EC3 can deploy and manage hybrid clusters across on-premises and public Cloud resources, thus introducing Cloud bursting capabilities. A case study is presented to assess the effectiveness of the tool featuring the structural dynamic analysis of buildings. In addition, checkpointing algorithms are evaluated in a real Cloud environment with existing workloads to study their effectiveness. The results show the feasibility and benefits of this type of clusters for computationally intensive applications.

Keywords: Cloud Computing, Checkpointing, Spot instances, Cluster Computing, Hybrid Clusters, Cloud Bursting

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

[☆]All the software presented in this work is available at `https://github.com/grycap`
[*]Corresponding author
Email address: `amcaar@i3m.upv.es` (Amanda Calatrava)

Throughput Computing (HTC). However, these computing platforms suffer from several drawbacks, such as the large upfront investment together with the maintenance cost that can suppose an important economic effort for small and medium-sized organisations. Moreover, the size of physical clusters cannot be easily adapted to the workload of the applications and they cannot offer customised environments for each application to be executed.

In the last years, the advent of hypervisors and virtualization technologies have paved the way for Cloud Computing. 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 [1]. Because of that, depending on the resource usage and the cost model, it might be convenient to deploy a virtual cluster instead of a physical one, as we concluded in a previous work [2]. Virtual clusters in the Cloud introduce profound benefits for many computational workloads, but specially for embarrassingly parallel jobs. These benefits include the on-demand provision of per-application customised clusters and the ability to dynamically increase and decrease the number of working nodes of the virtual cluster depending on the current workload, as we demonstrated in our earlier work [3]. This work resulted in the creation of EC3 (Elastic Cloud Computing Cluster)[1] [3] an open-source tool to deploy customised virtual elastic clusters on different on-premises, such as OpenNebula [4] and OpenStack [5], and public Cloud providers, such as Amazon Web Services (AWS) [6].

In this paper we build on our previous work to introduce two significant features: i) automatic checkpointing coupled with cost-effective mechanisms to provision transient computing capacity (commonly known as Spot Instances in AWS) and ii) the ability to deploy hybrid virtual clusters across on-premises and public Cloud platforms, featuring an elastic scheme that spans different Cloud providers.

Concerning the first feature, we need to benefit from cost-effective advantages offered by the cloud providers in order to reduce the total cost of the executions. This is the case of spot instances, a cloud pricing scheme available in Amazon EC2, where the users decide the maximum price they are willing to pay for the instance, thus offering up to 86% savings with respect to on-demand instances [7]. The user bids on spare Amazon EC2

[1]EC3: `http://www.grycap.upv.es/ec3`

instances and run them whenever the bid exceeds the current spot price, which varies in real-time based on supply and demand. This variation causes the instance to terminate if the spot price is higher than the bid of the user, thus causing the interruption of the job executions. This situation is known in the literature as an “out-of-bid” situation. Recently, Amazon has included spot instance termination notices [8], which provide a two-minute warning before the provider terminates the spot instance. Although this improvement is useful for some applications, two minutes is insufficient to checkpoint big applications, such as scientific applications, that might require additional time to save their context. Therefore, this type of instances offers lower cost at the expense of a reduced reliability. For that, checkpointing enables to periodically save the job progress before the spot instance is terminated by the provider, thus being able to resume from the latest checkpoint. In this work we review, propose and implement checkpointing algorithms for this purpose.

Concerning the second feature, to coexistence of on-premises and public Clouds has leveraged Cloud bursting, where virtual clusters can be enlarged with resources beyond the organisation, thus introducing the concept of hybrid clusters that can simultaneously harness on-premises and public Cloud resources. This introduces significant advantages for users to seamlessly access cluster-based computing resources beyond those available in their on-premises Clouds. Other topologies of hybrid clusters can be considered when using virtual resources, such as heterogeneous clusters, where different nodes of the cluster expose different hardware characteristics.

Therefore, this article extends the capabilities of EC3 to enable users to deploy self-managed cost-efficient virtual hybrid elastic clusters on top of Infrastructure as a Service (IaaS) Clouds. The remainder of this paper is structured as follows. First, section 2 covers the related work and the main contributions of this work to the state of the art. Next, section 3 focuses on the architecture and the new features included in EC3. Then, section 4 describes two case studies to assess the functionality and benefits introduced by the new features of EC3. Finally, section 5 summarises the main achievements of this paper and discusses the future work.

2. Related work

There are previous works in the literature that aim at deploying virtual clusters on Cloud infrastructures. For example, StarCluster [9] is an open-source tool to provision clusters in Amazon EC2 from a predefined configuration of applications (Open Grid Scheduler, OpenMPI, NFS, etc.). In this way, CycleCloud [10] is a commercial service provided by CycleComputing that deploys virtual clusters. However, both tools can only provision resources from Amazon EC2 and, therefore, virtual clusters cannot be deployed on on-premises Cloud platforms created with Cloud Management Platforms such as OpenNebula or OpenStack.

Elasticluster [11] can be employed to create virtual clusters on two Cloud providers (Amazon EC2 and Google Compute Engine) as well as on-premises Cloud platforms (OpenStack supported). The clusters can be scaled by the user and, thus, no automated elasticity is supported. Other tools to deploy virtual clusters can be found in the literature, such as ViteraaS [12], that allows the creation of virtual clusters to manage the execution of user-defined jobs, but users are not provided with direct access to the cluster. There are also commercial solutions, like IBM Platform Dynamic Cluster [13], that aims at partitioning on-premises 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-premises infrastructures and cannot be connected to commercial Cloud providers.

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 clusters over these 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-premises 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.

Regarding spot instances, many authors have made efforts developing predictive models for spot price variations. Some of the proposed solutions are based on Gaussian distributions [21] or Markov chains, like [22] and [23]. However, other authors found that the spot price variation in Amazon EC2 over time does not seem to follow any particular law [24]. It has also been observed in [25] that Amazon may be artificially intervening in the prices by setting a reserve price and generating prices at random, thus complicating even more the prediction of spot price variations.

Another field of research is the deployment of virtual clusters using spot instances. In [26] the authors consider the economics of purchasing resources on the spot market to deal with unexpected load peaks in a cluster, but they do not consider checkpointing techniques. If the instance is terminated, the application is restarted from the beginning. This is the case of [27] where the authors apply high bids rather than checkpointing strategies. This solution can incur in higher costs, loosing the main advantage of spot instances. Moreover, Amazon has recently limited [28] the bid of the user to ten times the on-demand price of the instance. Other solutions consider to deploy the cluster fully composed of spot instances [29] but again, they assume a bid value large enough to avoid the spot instance being killed by Amazon. Finally, the authors of [30] present SpotMPI, a toolkit to facilitate the execution of MPI applications on volatile auction-based cloud platforms. This toolkit can monitor spot instances and bidding prices, automate checkpointing at bidding price and automatically restart the application after out-of-bid failures. However, there are some limitations with this tool. First, it is based on StarCluster, so they are restricted to AWS. Second, the elasticity management of the clusters is not self-managed inside the cluster, because StarCluster implements the elasticity using the Elastic Load Balancer plugin [31], that runs on the local computer from which the cluster was deployed and requires permanent connection to the Cloud infrastructure to create and destroy the VMs. Instead, we propose self-managed virtual clusters where no external entities are required for elasticity management.

In conclusion, as far as the authors are concerned, there is no work in the literature that describes a tool that integrates hybrid virtual clusters combined with the use of spot instances and checkpointing techniques and that features self-managed elasticity. EC3 is provided both as a full-featured open-source development and a web-based interface that currently enables users to deploy their customised virtual clusters on AWS, OpenNebula and OpenStack.

3. Elastic Cloud Computing Cluster (EC3)

EC3 was developed to create virtual elastic clusters on top of IaaS Clouds. These self-managed clusters have the capability to adapt the size of the cluster to the workload, thus creating the illusion of a real cluster without requiring an investment beyond the actual usage. Therefore, they can scale out to a larger number of nodes (up to a maximum size specified by the user) depending on the number of jobs queued up at the Local Resource Management System (LRMS). Whenever idle resources are detected, the clusters dynamically and automatically scale in in order to cut down the costs in the case of using a public Cloud provider. The elasticity management is carried out by the front-end node of the cluster, with the help of CLUES [32]. More details about the elasticity management are available in [3].

EC3 supported different Cloud providers but the cluster had to be fully deployed in the same IaaS Cloud. Therefore, EC3 did not provide support for hybrid clusters. This paper describes the research and development carried out to introduce support virtual hybrid elastic clusters in EC3, where on-premises resources are supplemented with public Cloud resources to accelerate the execution process by provisioning an additional number of resources. Different instance types and the use of spot instances combined with on-demand resources are also cluster configurations supported by EC3.

Moreover, EC3 has been modified to let users improve the cost/performance ratio. On-demand instances involve higher costs than spot instances. However, the latter introduce higher risks at the expense of a lower cost and an increased provisioning time (delay until the bid is accepted). In order to alleviate the risks of using spot instances, such as the out-of-bid situation, EC3 uses checkpointing techniques. However, deciding when to perform a checkpoint to save the execution progress of the jobs running on spot instances is not a trivial task, specially in applications with large memory footprints, where the time to perform a checkpoint cannot be neglected.

It is important to point out that the user experience should be maintained regardless the physical location of the cluster and the type of machines that compose it. Indeed, the user should be unaware that the virtual cluster is actually composed of resources on top of one or more Clouds where the infrastructure dynamically adapts its size to the workload. The user is provided with the IP of the cluster and an SSH client is employed to access the front-end node.

The following subsections describe the overall architecture and its components.

3.1. Previously developed EC3 components

In order to deploy and configure the virtual cluster, we rely on previously developed components and open-source software available to the community:

- **RADL (Resource Application Description Language)** [33]: A declarative language for users to describe the computational infrastructure needed to run their applications.

- **VMRC (Virtual Machine image Repository and Catalog)** [34]: This component indexes Virtual Machine Images (VMIs) stored in different Cloud VMI repositories. It also implements matchmaking algorithms to obtain a ranked list of VMIs that satisfy the aforementioned given set of requirements (described in the RADL document).

- **CLUES (CLUster Energy Saving system)** [32]: This is an energy management system used in our architecture to manage the automatic elasticity of the virtual clusters. Initially created to work with physical clusters, it can work with virtual resources thanks to a cloud connector. It implements 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. More details about the elasticity policies can be found in [3].

- **Ansible** [35]: A DevOps tool to perform the unattended execution of commands specified in a YAML document in order to perform the automated installation of software dependencies. Therefore, this tool performs the installation of the required software packages so that the VMs behave as a cluster and the execution environment of the applications is successfully configured.

- **Infrastructure Manager (IM)** [36]: It is in charge of contacting different Cloud Management Platforms (CMPs) in order to deploy the VMs that compose the virtual cluster, whose requirements were described in the RADL document. It orchestrates also the contextualisation of those VMs by using Ansible. The IM uses a set of plugins that enables access to a large number of cloud deployments and virtualization platforms. It currently offers access to OpenNebula, OCCI, Amazon EC2, Google Cloud, Microsoft Azure, Docker, OpenStack and libvirt.

- **Local Resource Management System (LRMS)**: Two different LRMS are currently supported by EC3: SLURM and Torque + Maui. On the one hand, SLURM (Simple Linux Utility for Resource Management) [37] is an open-source resource manager that provides a framework for starting, executing, and monitoring work on a set of allocated nodes. On the other hand, Torque Resource Manager [38] provides control over batch jobs and distributed computing resources, and uses Maui as a job scheduler, to coordinate the execution of jobs over the cluster.

- **Berkeley Lab Checkpoint/Restart (BLCR)** [39]: A tool for transparently checkpointing applications, including MPI applications, where checkpoints are performed in an hybrid user/ kernel level. It does not require changes to be made to the application code to perform a checkpoint. It is used in EC3 to checkpoint applications running on spot instances, thus saving the job state.

- **Network File System (NFS)**: It is used to create a shared directory among the nodes of the cluster where the checkpoint files are saved, in a spot instance environment. Thus, when a spot node is destroyed, the checkpoint file of the job that it was executing can be accessed to restart the job in other node.

- **OpenVPN [40] and SSH tunnels**: These two technologies are used to allow connectivity between the front-end and the working nodes when some of the hosts are in a private network or behind a firewall. This situation typically arises in case of hybrid clusters.

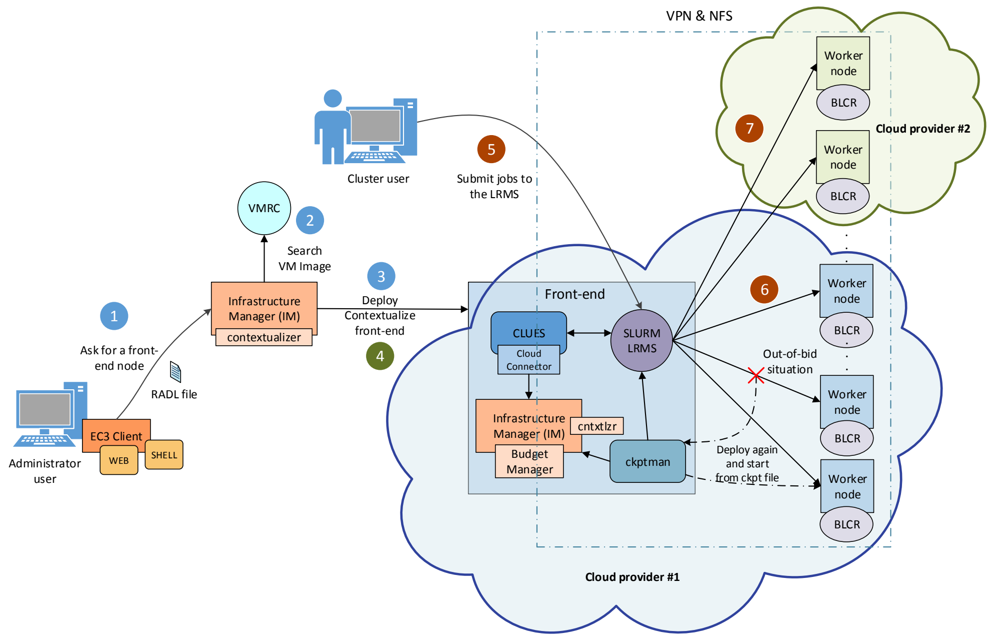


Figure 1: Proposed cluster deployment architecture. Phase 1 (blue), phase 2 (green) and phase 3 (brown) are also indicated.

3.2. Overall architecture

Figure 1 summarizes the main architecture of EC3. The deployment of the virtual cluster consists of three phases: (1) start a VM in the Cloud to act as the cluster front-end, (2) configure the front-end and (3) manage automatically the cluster size and configure new nodes depending on the workload.

Firstly, the user provides the EC3 client (using the GUI or the CLI) with the following data to perform phase (1): a cluster name, which facilitates the management of the cluster for the users (connecting via SSH, show information, reconfigure, etc.), the file that contains credentials to access the different Cloud providers and the end-point of the Infrastructure Manager to deploy the front-end together with the RADL files (step 1 in the Figure 1). Here the user can use the default RADLs provided by the tool (such as the necessary instructions to configure a SLURM cluster or the hybrid features), but also, the user can add additional customised RADLs to configure the cluster with specific applications. Inside the RADLs, the characteristics of the nodes are given in terms of hardware, software and configuration requirements.

Moreover, the user can specify the maximum number of nodes that compose the cluster. It is also supported the specification of a maximum number of nodes from a given type (for example, the maximum number of nodes based on spot instances or the maximum number of on-demand nodes). The user can change these values during the lifecycle of the cluster by using the `reconfigure` command of the EC3 client.

With the data specified by the user, EC3 contacts the IM to deploy the front-end. The IM first selects the appropriate VMI for the front-end. It can take a particular user-specified VMI, or it can contact the VMRC to choose the most appropriate VMI available (step 2), considering the requirements specified in the RADL file. Then, the IM chooses the IaaS Cloud provider, and the type of instance (spot or on-demand if necessary) according to the requirements of the user (step 3). The user can specify in the RADL file whether spot instances will be used or not. Notice that the IM will only use spot instances if the actual spot price is lower than the on-demand price. Finally, phase 1 concludes deploying an instance of that will be used as the front-end node of the cluster.

Phase 2 starts once the aforementioned instance is available, installing and configuring all the required software that is not already preinstalled in the VM (step 4). In this phase, all the required software is installed to configure the instance as the front-end of the cluster. This involves deploying: i) a new IM in the front-end that will be used to deploy the worker nodes; ii) Ansible to configure the new nodes; iii) CLUES, in charge of managing the elasticity of the cluster; iv) the LRMS selected by the user and v) (optionally) additional software packages specified by the user. If the cluster is configured to be hybrid, VPN (Virtual Private Network) or SSH tunnels will be configured to interconnect all the nodes (eligible by the user). Moreover BLCR and NFS will be also configured if the user enables the use of spot instances.

Once the front-end has been deployed and configured, phase 3 starts. At this moment, the virtual cluster (composed only of the front-end node) becomes totally autonomous and users 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 (step 5). The user will have the illusion of a cluster with the number of nodes specified as maximum size. CLUES will monitor the working nodes and intercept the job submissions when 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. Just like in the deployment of the front-end, CLUES internally uses the IM configured in the front-end in phase 2 to deploy additional VMs that will be used as working nodes for the cluster (step 6). Once these nodes are available, they are automatically integrated in the cluster as new available nodes for the LRMS.

Finally, step 7 represents a hybrid situation, where some nodes of the cluster are deployed in another Cloud provider to satisfy the requirements of the user. More details about hybrid cluster configurations will be analysed in section 3.4.

3.3. Checkpointing Manager (ckptman)

Ckptman is a tool specifically developed for EC3 that automates checkpointing of the jobs running on spot instances in order to save as much job execution progress (and reduce cost) as possible. It is created to work with Amazon spot instances and the BLCR checkpointing tool, within the EC3 cluster deployment tool. It also uses NFS to share the directory where the checkpoint files are saved. This way, when a spot instance is terminated by Amazon, the checkpoint file is not lost and the job can be restarted in another node. Notice that the front-end node of the cluster is never deployed as a spot instance.

The structure of ckptman is described in Figure 2. The tool, deployed in the front-end, consists of a daemon that checks every preset amount of time the state of the jobs running in the nodes deployed using spot instances. The IM connector gets which nodes were deployed using spot instances from Infrastructure Manager (IM). Also it gets the state of the jobs from the LRMS. Depending on the algorithm chosen from the ones described in section 3.3.1, it decides if it is necessary to perform a checkpoint or not. Checkpoints are taken by invoking BLCR commands using SSH connections to the node where the checkpoint must be taken. Notice that BLCR supports checkpointing MPI processes, so parallel jobs can also be checkpointed by ckptman.

When a node is destroyed by Amazon due to an out-of-bid situation, ckptman notices this fact and enqueues the affected job again, recovering the state of the job from its last checkpoint, or from the beginning if no checkpoint file is found for that job. Notice that this might cause the deployment of a new node if no idle nodes are available in

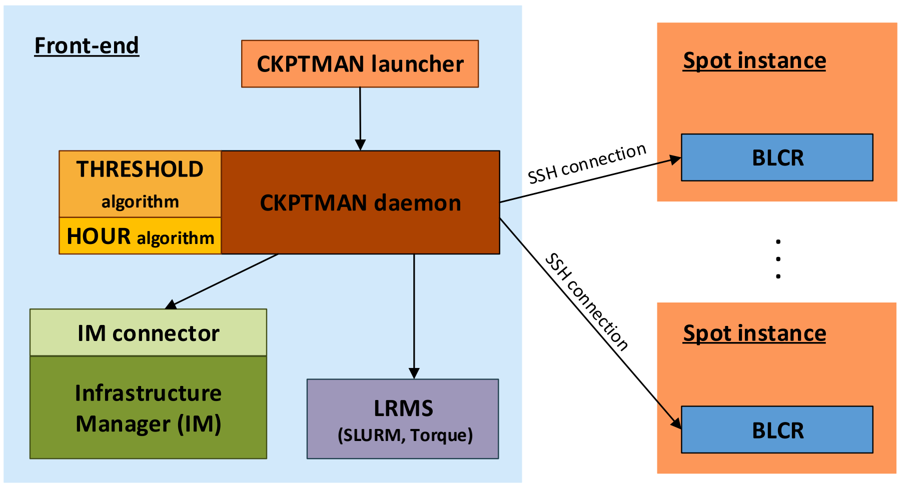


Figure 2: The structure of ckptman inside the front-end of the cluster and the relation with spot nodes.

the cluster. The next subsection analyses the checkpointing algorithms described in the literature and justifies the algorithms implemented in ckptman.

3.3.1. Checkpointing algorithms

The literature already includes several checkpointing strategies in order to save the work just before an out-of-bid situation occurs, like [41] or [42]. However, none of them are actually tested in a real environment and, thus, the authors do not describe the technologies they are using to perform the checkpoints. In fact, the results presented in these works are based on simulations. Even some of them consider infinite values of bid [43], thus making checkpointing unnecessary. Therefore, we describe both the implementation of one of these solutions and a proposal of a new algorithm together with their assessments in real scenarios to evaluate their performance.

After analysing the checkpointing strategies described in the literature with the performance obtained in the simulations performed by the authors, we have chosen two different checkpointing algorithms to be implemented in our tool:

- **Hourly Checkpointing (`HOUR`):** checkpoints are taken just prior to the beginning of the next instance hour, based on a checkpoint margin time. Since Amazon does not charge any partial hour when it terminates the instances due to an out-of-bid situation, this algorithm, firstly proposed in [41], will save the job process that the user has already payed. The margin time to perform a checkpoint can be easily adapted to the time needed to perform the checkpoint to the user application, which depends on the application memory consumption and other factors, like the MPI communications used.

- **Threshold Checkpointing algorithm (`THRESHOLD`):** checkpoints are taken when a rise in the price of the spot instance is observed within an interval. The lower limit of the interval is a fraction of the price the user has determined. This value is recalculated every 10 minutes or when a checkpoint is taken. The upper limit is the bid of the user. Moreover, checkpoints are also taken every hour.

  The lower limit of the interval is:

  $$Threshold = \frac{P_{\overline{x}} + U_{bid}}{2}$$

  where $P_{\overline{x}}$ is the average of the prices in the last period of 10 minutes, and U_{bid} represents the bid of the spot price request.

  However, it is necessary to avoid the overload of the system caused by massive checkpointing operations in fluctuating periods over the upper limit. For that, when a checkpoint is taken, the lower limit of the threshold is recalculated by:

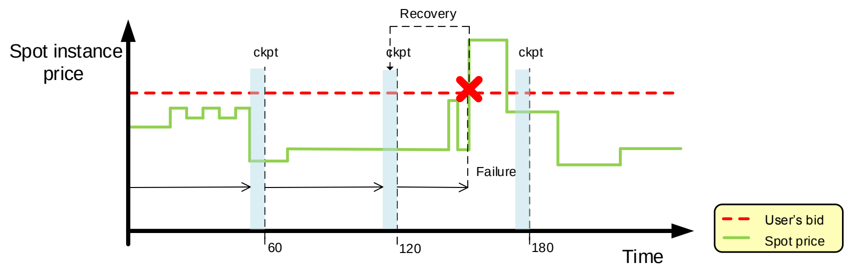
  $$Threshold_{ckpt} = \frac{P_{ckpt} + U_{bid}}{2}$$

  where P_{ckpt} represents the price that has caused the checkpointing operation. This formula is also applied when the virtual machine is started, to calculate the initial threshold, where the value of P_{ckpt} represents the price when the instance was deployed.

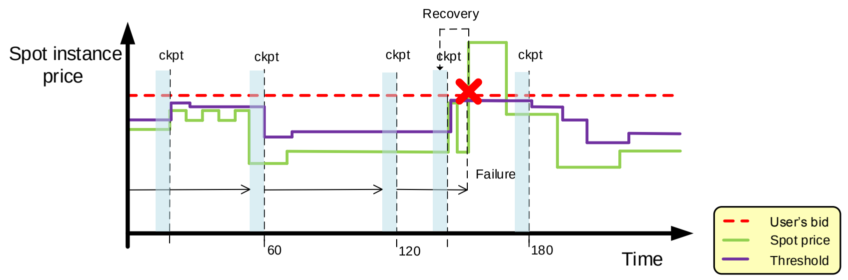
  It can be considered as an evolution of the algorithm proposed in [42], for two main reasons. Firstly, the hourly checkpoints can save job process facing an unexpected peak where the variation of prices is very high. Secondly, the new way to calculate the threshold adapts better its value avoiding multiple checkpoints in a variable period near the user's bid. Algorithm 1 represents the threshold algorithm in a more programmatic way.

Algorithm 1 Threshold algorithm

```
Require: launch time, $launch\_time$, of the node; user's bid, $u\_bid$; checkpoint time margin, $m$
  $checkpoint$ = false
  Obtain actual time, $actual\_time$
  $life\_time$ = $actual\_time$ - $launch\_time$
  $remaining\_hour\_time$ = 3600 - $life\_time$ % 3600
  {Checkpoint if time is close to the next instance hour}
  if $remaining\_hour\_time < m$ then
    $checkpoint$ = true
  end if
  Obtain historic prices in the last 10 minutes from Amazon EC2, $p$
  $threshold = (\bar{p} + u\_bid)/2$
  {Checkpoint if the most recent price is greater than threshold}
  if $p[0] > threshold$ then
    $checkpoint$ = true
  end if
  if $checkpoint$ then
    $threshold = (p_l + u\_bid)/2$
  end if
  return $checkpoint$
```



(a) Behaviour of the `HOUR` checkpointing algorithm.



(b) Behaviour of the `THRESHOLD` checkpointing algorithm.

Figure 3: Behaviour of the checkpoint algorithms implemented in ckptman in a simulated scenario.

Figure 3 shows an example of the behaviour of the algorithms implemented in ckptman for the same variation of spot prices. When an out-of-bid situation occurs, both algorithms are able to recover the job execution progress but from different points. The `THRESHOLD` algorithm loses less job execution progress than the `HOUR` algorithm, but it also makes more checkpoint operations, although the threshold is recalculated to adapt it to the price variations. Notice that additional checkpointing algorithms can be developed and easily introduced in ckptman[2].

3.4. Hybrid features

One of the objectives of the new developments added to EC3 is to ease the deployment and management of virtual hybrid clusters whose computational resources are si-

[2]The source-code of ckptman is available at: `https://github.com/grycap/ckptman`

multaneously provisioned from on-premises Clouds and from different public IaaS Cloud providers. This is of special interest to introduce Cloud bursting and enable users to seamlessly access a larger amount of computing power.

Provisioning nodes across multiple Clouds with private IP addresses hinders connectivity among nodes on different clouds. Therefore, EC3 deploys a VPN server on the front-end to which the worker nodes automatically connect. In case the front-end is behind a firewall that disallows port forwarding, we also allow to deploy reverse SSH tunnels on the affected nodes.

In spite of the interconnection strategy chosen, EC3 can create four different types of hybrid cluster configurations:

- **On-premises resources + public resources**: This is the most clear example of Cloud Bursting. The cluster is composed of virtual nodes, deployed inside the on-premises Cloud of the organization. When the cluster needs to grow and there are no more resources available inside the on-premises Cloud (because of user quotas or if the infrastructure is overloaded), the new nodes are deployed in a public Cloud. Notice that the nodes provisioned from the public Cloud can be on-demand or spot.

- **On-demand resources + spot instances**: The cluster is deployed in a public Cloud where nodes can be on-demand or spot. Spot nodes should have checkpointing capabilities in order to save the job execution progress in an out-of-bid situation.

- **Different instance types**: the cluster can be composed of nodes with different characteristics (typically, CPU and RAM), conforming an heterogeneous cluster. For example, in AWS, a cluster can be composed of small (1 vCPU and 2 GiB of RAM)[3] and medium (2 vCPU and 4 GiB of RAM) instances. This type of configuration is of special interest for heterogeneous parallel computing. Notice that the software configuration can be the same for all the nodes that compose the cluster, but it is also possible to configure specific software packages for a particular type of node.

[3]Each vCPU is a hyperthread of an Intel Xeon core.

- **Different public or private Cloud providers**: the cluster is deployed across different Cloud providers, provisioning resources from multiple Clouds.

Notice that all possible combinations from the four types above, are also supported by EC3. HPC applications should take special consideration when they run in hybrid scenarios. If all nodes running a job are not in the same Cloud or datacenter the performance can be compromised by high latency or low bandwidth in accessing the shared file system and performing explicit communications (e.g. as it happens for MPI applications). We prevent this by creating a nodes partition per Cloud in the LRMS. For example, SLURM will enforce assigning all the nodes for job from the same partition. However we let the LRMS deal with the possible fragmentation of resources, i.e., the phenomenon in which there are idle nodes that cannot be assigned to a job because they are not on the same partition.

4. Case studies

In this section two case studies. The first one is presented in subsection 4.1 to assess the effectiveness of the tool by means of a computationally intensive scientific application that performs the structural dynamic analysis of buildings. The results show the feasibility and benefits of this type of clusters for computationally intensive applications. Moreover, checkpointing algorithms are evaluated in a real environment with real workloads to study their effectiveness, where the main results are presented in subsection 4.2.

4.1. Structural Dynamic Analysis of Buildings using EC3

In order to assess the effectiveness of a self-managed cost-efficient virtual hybrid elastic cluster on a Cloud infrastructure, we present a structural dynamic analysis of buildings use case. The structural dynamic analysis of buildings is required to accurately simulate how a building is affected by external dynamic loads, such as an earthquake. This is a computing-intensive task that can be efficiently tackled with parallel computing on clusters of PCs. Simulations will be executed by means of the structural simulator component of the Architrave software [44]. It consists of a MPI-based HPC batch application that can be benefit from on-demand clusters provisioned from the Cloud in order

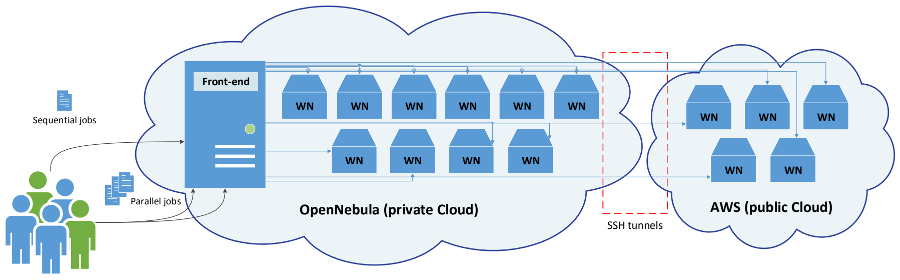


Figure 4: Scenario of the first case study: A hybrid virtual elastic cluster across an on-premises (OpenNebula) and public Cloud (AWS).

to offer an online service for structural analysis to a community of users. This structural simulator has some library dependencies, such as PETSc or SLEPc.

The case study involves a virtual infrastructure specially dedicated to perform structural dynamic analysis of buildings, where multiple users can access and execute their jobs. A simplified representation of the scenario is shown in Figure 4. We are going to perform two different executions, over two distinct cluster configurations, (scenario *a*) a hybrid infrastructure composed of nodes from our on-premises Cloud and public nodes from AWS, only considering on-demand instances, and (scenario *b*), a hybrid infrastructure composed of nodes from our on-premises Cloud and public nodes from Amazon EC2, using spot instances.

The job pattern submission used in both cases is represented in Figure 5, and it has been specially designed to test the infrastructure elasticity and its ability to create hybrid clusters. The jobs can be sequential (HTC) or parallel (HPC), so they might need more than one node, and also have different duration, thus considering the job heterogeneity that physical clusters cope with. Specifically, the case study is composed of four types of jobs: i) sequential jobs with an approximate duration of one hour in a single node, ii) parallel jobs with an approximate duration of one hour in two nodes, iii) sequential jobs with an approximate duration of two hours in a single node, and iv) parallel jobs with an approximate duration of two hours in two nodes. Notice that, as it has been said before, jobs that require more than one node are executed entirely by nodes of the same Cloud by using SLURM partitions that represent each Cloud, thus ensuring high performance and low latency communications among those nodes. A total of 124 jobs is executed in

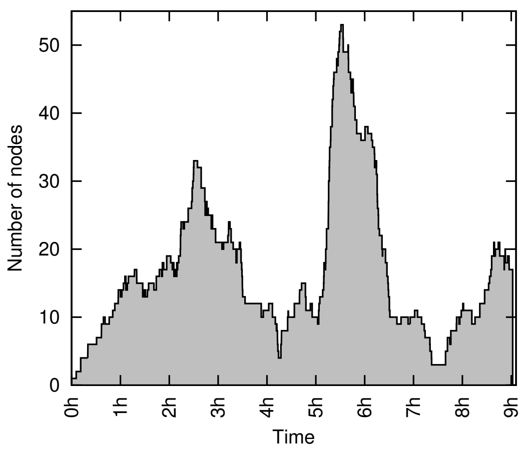


Figure 5: Job pattern submission during the case study. A total of 124 jobs are executed.

each execution, where 83 are sequential and 41 are parallel ones.

The infrastructure used to deploy the on-premises Cloud is composed of 8 dual processor with 14 core nodes (28 cores per node), with 64 GiB of RAM and a shared storage system of 10 TiB, backed by a Storage Area Network (SAN) where the hard disks are stored as volumes. This system is managed by OpenNebula 4.8, using KVM as the underlying hypervisor. With respect to the public Cloud, we rely on Amazon Web Services. The instance type chosen is `m1.medium`, with one (virtual) processor, 3.75 GiB of RAM and 410 GiB of disk. The Virtual Machine Image (VMI) used in the on-premises Cloud provides the same execution environment as the EC2 AMI (Amazon Machine Images) employed, an Ubuntu 12.04 LTS with BLCR and SLURM preconfigured. In this way, all VMs deployed on both Clouds have the same characteristics. We have chosen a preconfigured VMI to accelerate the deployment time of the working nodes, as we explain later, but it is also possible to configure all this software on-demand. For both executions we use SSH tunnels to interconnect the nodes.

Moreover, we fixed a limit of 50 nodes (plus the front-end) for the size of the cluster. A maximum of 35 of these nodes can be from the on-premises Cloud, and the remainder (15 nodes) were from the public Cloud.

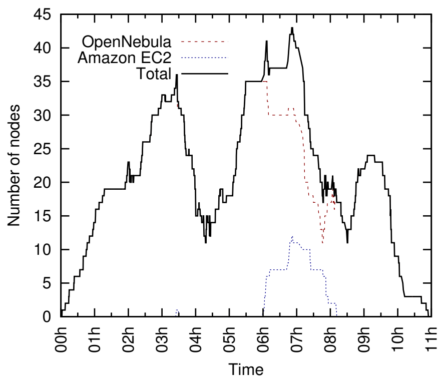
	OpenNebula instance	EC2 On-demand instance	EC2 Spot instance
Deployment	40	45	335
non-preconfigured VMI:			
Contextualisation	661	698	702
Total avg. time	701	743	1037
preconfigured VMI:			
Contextualisation	196	232	337
Total avg. time	236	277	672

Table 1: Deployment and contextualisation average time (in seconds) for nodes deployed in different Clouds.

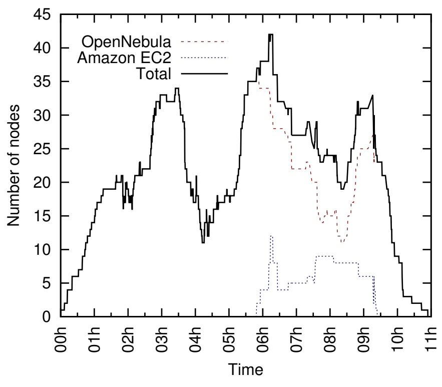
4.1.1. Results and Discussion

First, we are going to analyse time differences in the deployment and contextualisation processes for the types of virtual machines that we are using in our case study. Table 1 shows the average time obtained for each step for VMs deployed in our on-premises Cloud (second column), on-demand instances deployed in Amazon EC2 (third column) and spot instances from EC2 (fourth column). As it can be observed, the deployment time in case of using spot instances is considerably higher than with on-demand instances, due to the time invested by AWS to fulfil the spot request (335 seconds in average). Also, we can see the differences in the contextualisation process depending on the use of preconfigured VMIs (with only BLCR and SLURM pre-installed). Because of these differences, we decided to use a preconfigured VMI in the subsequent executions. Notice that we still need to configure SLURM configuration files, SSH tunnels, the NFS system and the application dependencies. However, starting from a VMI with BLCR and SLURM previously installed significantly reduces the time consumption in the contextualisation process.

Second, we present the results of the scenario *a* and scenario *b* executions using the job pattern submission represented in Figure 5. For both executions we used conservative elasticity policies to ensure the minimum cost of the infrastructure, in terms of energy consumption (for the on-premises Cloud) and budget consumption (for the public Cloud). Thus, the *scale out* policy selected will deploy a new node (or two nodes if the job requires them, like parallel jobs in our case) only if there are no idle nodes in the virtual cluster. EC3 will deploy nodes in the on-premises Cloud until it reaches the limit (35 nodes). If more nodes are required, they will be deployed in the public Cloud. The *scale in* policy will remove a node from the infrastructure when it is idle during 60 seconds and no more jobs are queued up at the LRMS. Notice that it is possible to select different scale in and out policies that EC3 offers, like group-based start of nodes or time blocks to destroy them, but in this case study we want focus on the hybrid features and the cost savings provided by the use of spot instances together with checkpointing techniques. More details about the elasticity policies can be found in [3].



(a) Hybrid infrastructure using on-demand instances.



(b) Hybrid infrastructure using Spot instances.

Figure 6: Behaviour of the Virtual Hybrid Elastic Cluster in both executions.

In the first execution, represented in Figure 6(a), a hybrid infrastructure composed of nodes from our on-premises Cloud (OpenNebula) and on-demand public nodes from AWS has coped with the execution of jobs. Notice that three periods of scale out and three periods of scale in can be observed. An accurate analysis of the results shows two overload periods of the on-premises Cloud. The first one occurs at 3h 20 min, where the on-premises Cloud has 35 nodes deployed and there is at least one job pending in the queue. EC3 decides to deploy a new node in the public Cloud, but what we can see in the graph is that before the node deployed in EC2 is ready, a node in our private Cloud finishes the execution of its jobs and automatically the pending job in the queue is submitted to that node by the LRMS. Since EC3 did not receive new jobs during that period, it terminated the instance provisioned from AWS.

The second important point in the graph starts at 5h 30 min, where the frequency of job submission is less than a minute, causing the saturation of the infrastructure. Since all 35 nodes deployed in OpenNebula are executing jobs, EC3 starts to provision new nodes in AWS to execute the new jobs. At 6h, working nodes deployed in OpenNebula finalise their execution and start to receive new jobs, thus helping to solve the saturation situation. Note that the deployment and contextualisation of new working nodes is carried out in parallel.

Figure 6(b) represents the results of the second execution, where EC3 has configured a hybrid infrastructure composed of nodes from our on-premises Cloud and nodes from AWS, using spot instances. The checkpoint algorithm chosen for this execution was *threshold*. We consider a basic opportunistic approach, to set the maximum bid price for the spot requests to the on-demand price for the same type of instance, 0.087\$. With that strategy, we ensure to pay for a spot instance no more than the price for an on-demand instance, since AWS guarantees that you will never pay more than your maximum bid price per hour. Notice also that we might pay less per hour than our maximum bid price, due to the periodical adjusts in price that AWS performs, where everyone pays the same spot price for that period regardless of whether their maximum bid price was higher.

As we discussed before with Table 1, working nodes deployed with spot instances need more time to be ready for the execution of jobs. This can explain the main differences observed between Figure 6(a) and Figure 6(b). Regarding checkpointing, a total of 29 checkpoints with an average duration of 160 seconds, have been completed during the execution. These is an example where the two minutes warning added recently by Amazon to inform about the close termination of the spot instance is not enough. It is necessary to point out that the IM chooses the cheapest availability zone for the spot instances request, in this case *us-east-1e*. As it is represented in Figure 7, this region does not suffer price variations during the execution, so the checkpoints taken correspond to the hourly ones.

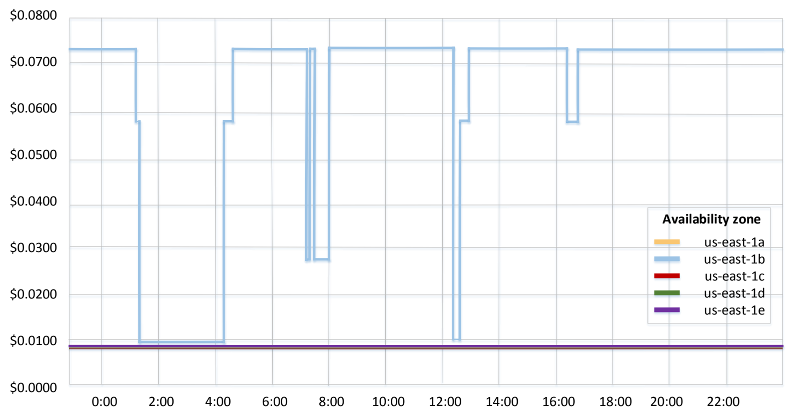


Figure 7: Spot price history corresponding to 8th April 2015, when the case study was executed, for *m1.medium* instance type.

	On-demand execution	Spot execution
Seq. jobs avg. waiting time	7min 14s (434s)	10min 9s (609s)
Par. jobs avg. waiting time	14min 43s (883s)	17min 56s (1076s)
Total time of execution	10h 47min 12s (38832s)	10h 56min 21s (39381s)
Total cost of execution	2.349$	0.234$

Table 2: Details of time and price about the executions.

Table 2 details the time and the cost involved in both executions. It can be noticed that the total time of execution is similar in both executions, but the cost is sensitively lower when we use spot instances, as we expected. The actual on-demand price for a *m1.medium* instance is 0.087$, while the spot price for the same instance during our execution was 0.0081$. Thus, the total cost of the execution using spot instances is a 10% of the execution with on-demand instances. Also, the differences in the job waiting time in the queue between sequential and parallel jobs is caused by the scheduling policy selected. When a new job arrives to the queue asking for two nodes (or more), CLUES automatically deploys the number of nodes requested by the job via the IM (if there are no available nodes in the cluster). If a second job arrives to the LRMS queue during the deployment process requesting less nodes than the first job, it will probably start its execution earlier than the first job. This is because the deployment and contextualisation

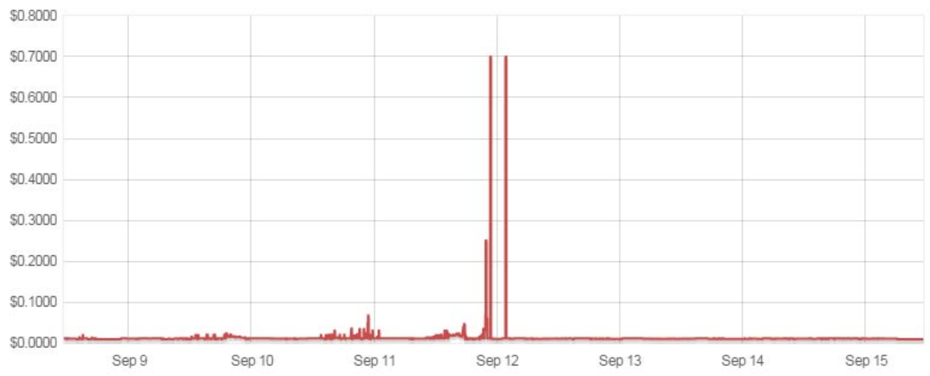


Figure 8: Spot price history corresponding to the period from 9th September 2015 to 15th September 2015, for *m3.medium* instance type in the availability zone *us-east-1c*. The OS is *Linux/UNIX (Amazon VPC)*.

process of the new nodes do not finalize at the same time, usually with a few seconds of difference between nodes launched at the same time, but enough to detect an active node before the others. This causes that the LRMS detects an active node and submits the second job to it. CLUES can be also configured to behave in a different way, and wait for the second node to be active without starting a new job in the first node.

4.2. Analysis of the checkpointing algorithms

In this subsection we provide a detailed study of the effectiveness of the two checkpoint algorithms, in charge of deciding the best moment to checkpoint an application running on a spot instance. We base our analysis on real workloads obtained form the Grid Workloads Archive [45], with the aim of testing the developed algorithms in a real environment, thus differentiating our work from most of related works that can be found in the literature, which are typically based on simulations.

In order to rigorously compare the behaviour and performance of the two checkpointing algorithms proposed in this work (`HOUR` and `THRESHOLD`), we are going to perform three different executions under a spot environment. The first one will not use checkpointing techniques (`NONE`). The second one will use the `HOUR` algorithm and the third one will use the `THRESHOLD` algorithm. This means that we need to use the same price variations for all the executions, in order to properly compare the results. For that,

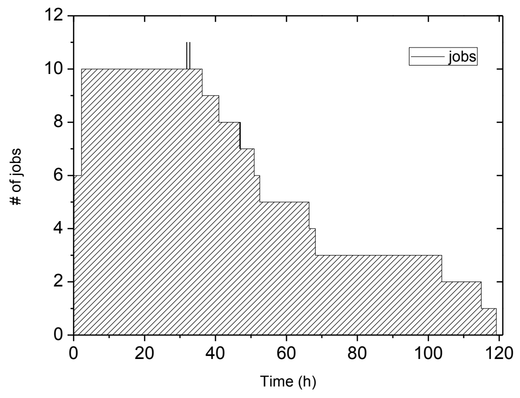


Figure 9: Workload of the case study, extracted from the *GWA-T-3 NorduGrid* dataset.

we have selected a specific fragment of the spot price history offered by Amazon EC2, illustrated in Figure 8. The period used in the case study is from September 9, 2015 to September 14, 2015, for *m3.medium* instance type in the availability zone *us-east-1c*. The user bid will be 0.067$, the on-demand price for this instance type, using the same opportunistic approach as in the first case study. Finally, only for comparison purposes, the fourth execution will be an on-demand execution, using the same type of instance, *m3.medium*.

As the objective of this case study is to analyse the efficiency of the checkpointing algorithms under a spot scenario, the configuration of the cluster is an on-demand *m3.large* front-end and spot *m3.medium* nodes, all of them deployed in AWS. The maximum size of the cluster has been fixed to 12 nodes, according to the workload used. This real workload is a fragment extracted from the *GWA-T-3 NorduGrid* dataset offered by the Grid Workloads Archive (from line 4 to line 16 of the *.gwf* file), and represented in Figure 9 in terms of size evolution of the cluster. The jobs executed in that dataset are sequential [46]. More parameters and values of this case study are presented in Table 3.

Notice that the recovery time of the jobs depends on if there is an available node to execute the re-submission of the job or not. If a node exists, an average of 74 seconds is required to recover a job from its checkpoint. However, if it doesn't exist, EC3 needs to deploy a new node to execute the job, thus consuming the time needed to deploy and configure this new node plus the detection and re-submission of the job (an average of

Task avg. time	191407s
Node deployment avg. time	672s
Checkpoint avg. time	160s
Recovery avg. time	[74, 746]s
User bid	0.067€
ckptman revalue time	30s

Table 3: Parameters and values of the second case study.

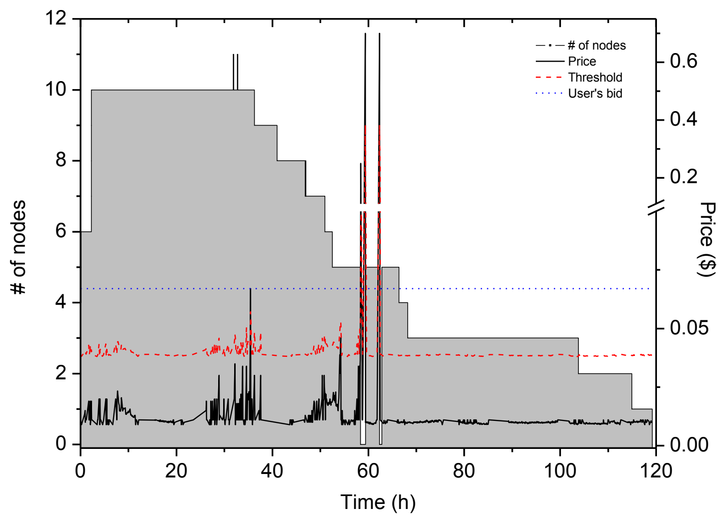
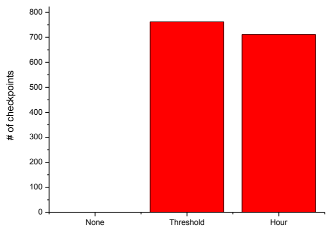


Figure 10: Threshold evolution of the **THRESHOLD** algorithm vs price history of *m3.medium* instance type. Also the evolution of the cluster in terms of nodes is represented.

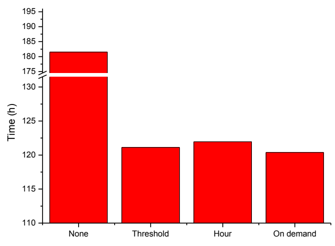
746 seconds). It will happen when a high increase in the spot price occurs, exceeding the user's bid, because all the nodes will be killed by Amazon due to this out of bid situation. The *ckptman* revalue time offered in Table 3 stands for a configurable parameter that indicates how often the jobs and nodes are reevaluated by *ckptman* to consider if it is necessary or not perform a checkpoint.

4.2.1. Results and Discussion

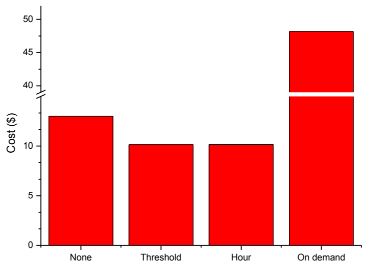
In this subsection we are going to analyze and discuss the results obtained from the four executions proposed above. Firstly, Figure 10 represents the threshold evolution of the THRESHOLD algorithm and the price history of *m3.medium* instance type during the THRESHOLD execution. The evolution of the size of the cluster in terms of nodes is also represented. We can extract from this graph a general view of the scenario of the case study, applicable for the three spot executions performed. On the one hand, we can easily observe that three important increases in the spot price will destroy our working nodes of the cluster (between hours 57 and 63), due to an out-of-bid situation. Five jobs will be affected, and they will need to be restarted. On the other hand, we can notice specifically for the THRESHOLD execution, how the value of the threshold is continuously adapted to the price evolution. This way, a checkpoint was taken due to the price increase in the hour 36. However, the rises during hours 57 and 63 are such high that the virtual machines are killed before the algorithm can perform the checkpoint corresponding to the increase. However, notice that the THRESHOLD algorithm also takes checkpoints every hour, to reduce the loss in job execution progress.



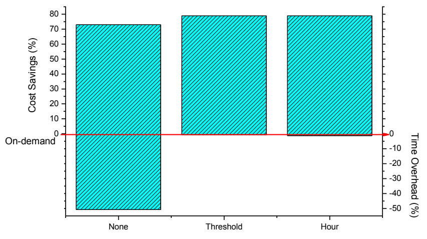
(a) Number of checkpoints per algorithm.



(b) Total execution time.



(c) Total costs of the execution.



(d) Percentages of cost saving and time overhead with respect to an on-demand execution.

Figure 11: Performance evaluation of the three checkpointing strategies.

To better analyze the four executions performed, we present Figure 11, where several graphs compare them in terms of number of checkpoints, total time and total costs. From Figure 11(a), we can find the number of checkpoints performed by our algorithms during the execution. A difference of 7% can be observed between `HOUR` and `THRESHOLD` algorithms. Obviously neither `NONE` nor on-demand executions performed a checkpoint. Figure 11(b) represents the total execution time for the four executions. As it was expected, the on-demand execution is faster than the spot executions. Also, the differences between performing or not checkpoints during the spot execution are considerably high. Indeed, the `NONE` execution supposes more than a 50% of time overhead comparing to the on-demand execution. Nevertheless, `HOUR` and `THRESHOLD` executions suppose only an overhead of 1.3% and 0.63% regarding the on-demand execution, respectively. The next graph shows the total cost of the executions represented in Figure 11(c). The major costs are caused by the on-demand execution (48,17$), as it was expected. Then, the `NONE` execution is the second more expensive (13.02$) and finally, the `HOUR` and `THRESHOLD` executions have a similar cost (10.16$ and 10.15$ respectively)). Finally, Figure 11(d) represents the performance of the studied algorithms in terms of costs and time with respect to the on-demand execution. The overhead time represents the difference between the time of the on-demand execution and the spot execution. The best results are performed by the `THRESHOLD` algorithm, with a 78,94% of savings and an overhead time of 0.63% with respect to the on-demand execution. However, the `HOUR` algorithm also obtains good results, with a 78,92% of savings and a time overhead of 1.3%.

From this data analysis of the executions, we can emphasize two important conclusions. On the one hand, the importance of using checkpointing techniques when a spot execution wants to be performed. An spot execution can reduce considerably the costs but it can also increase significantly the total time of the execution. This increment can be mitigated by using checkpointing techniques, a conclusion achieved in other works of the literature, such as [41] and [43]. On the other hand, comparing both algorithms presented in this paper (`HOUR` and `THRESHOLD`) we can conclude that the `THRESHOLD` algorithm is able to adjust more the costs and time overhead, but its performance is highly related to the price history variations. If the increment that causes an out-of-bid situation occurs without a predecessor value inside the `THRESHOLD` interval that does not provoke an out-of-bid situation, the algorithm cannot predict this rise. In that situation, the job execution progress saved relies on the checkpoint performed in the last hour. For that reason, the differences between them are not very significant in our case study. However, with a spot price history that presents continued fluctuations before the out of bid situation, the `THRESHOLD` algorithm is expected to increment its performance. In this type of scenarios, the use of the `THRESHOLD` algorithm is recommended. At the same time, the usage of the `HOUR` algorithm is recommended for situations where the spot price remains steady until a high increase in the spot price occurs, thus causing an out-of-bid situation. Also, considering that partial hours are not charged for terminated spot instances, the use of the `HOUR` algorithm in this type of situations is reinforced. This conclusion supports the ones presented in [41].

5. Conclusions and future work

This paper has described the developments in Elastic Cloud Computing Cluster (EC3), a tool that creates self-managed cost-efficient virtual hybrid elastic clusters out of computational resources provisioned from multiple IaaS Clouds. In particular, the paper has focused on two topics: i) hybrid clusters across on-premises and public Clouds and ii) leveraging spot instances, offered by AWS, to introduce reliable low cost cluster-based computing. A case study was executed that involved a scientific application to perform the nonlinear dynamic analysis of buildings executed in a hybrid virtual elastic cluster across an on-premises OpenNebula Cloud and AWS. Also, the threshold algorithm was introduced and assessed together with other algorithms in the literature using real workloads and real spot prices.

The results show the ability of the clusters to adapt their size to the workload, the automatic Cloud bursting to a public Cloud, and the significant savings that suppose the use of spot instances in contrast with on-demand instances, with the increased resilience that arises from periodic and automatic checkpointing of the jobs. EC3 is open-source, based on the Apache 2.0 License, and hosted on GitHub[4] and a web application is also offered for free (Elastic Virtual Clusters as a Service) to the community[5].

[4]EC3 at GitHub: `https://github.com/grycap/ec3`

[5]EC3 web GUI, available at `http://www.grycap.upv.es/ec3`

Future work involves adding migration capabilities to EC3. We want to enable the virtual clusters to be migrated across Cloud platforms, thus introducing an unprecedented degree of flexibility for datacenters, specially during planned outages where computing power can be temporarily outsourced to a public Cloud. Besides, we will consider to address the checkpointing of applications that require also restoring the content of the file system at the moment of the checkpointing. We also plan to adapt the algorithms to other public Cloud providers with similar features. This is the case of the preemptible VM instances provided by the Google Cloud Platform.

Acknowledgments

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 form The Spanish Ministry of Economy and Competitiveness to develop the project "CLUVIEM", with reference TIN2013-44390-R. Finally, authors wish to express their gratitude to D. David Ruzafa, for his support in the arduous task of analyzing the data of the executions.

[1] P. Mell, T. Grance, The NIST definition of cloud computing, Tech. Rep. 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD (September 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 Gener. Comput. Syst. 29 (3) (2013) 704–712. `doi:10.1016/j.future.2012.08.014`.
URL `http://dx.doi.org/10.1016/j.future.2012.08.014`

[3] M. Caballer, C. De Alfonso, F. Alvarruiz, G. Moltó, Ec3: Elastic cloud computing cluster, J. Comput. Syst. Sci. 79 (8) (2013) 1341–1351. `doi:10.1016/j.jcss.2013.06.005`.
URL `http://dx.doi.org/10.1016/j.jcss.2013.06.005`

[4] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, Capacity Leasing in Cloud Systems using the OpenNebula Engine, Cloud Computing and Applications 2008 (CCA08), 2009.

[5] OpenStack, OpenStack Cloud Software, `http://openstack.org`, [Online; accessed 1-December-2014].

[6] Amazon, Amazon Web Services (AWS), `http://aws.amazon.com`, [Online; accessed 1-December-2014].

[7] Amazon, Spot instances (amazon ec2), `https://aws.amazon.com/ec2/purchasing-options/spot-instances/`, [Online; accessed 1-December-2014].

[8] A. EC2, Spot instance termination notices, `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html`, [Online; accessed 13-January-2015].

[9] MIT, Starcluster, `http://web.mit.edu/stardev/cluster/`, [Online; accessed 1-December-2014].

[10] CycleComputing, Cyclecloud, `http://www.cyclecomputing.com/cyclecloud`, [Online; accessed 10-December-2014].

[11] U. of Zurich, Elasticluster.
URL `http://gc3-uzh-ch.github.io/elasticluster/`

[12] F. Doelitzscher, M. Held, C. Reich, A. Sulistio, Viteraas: Virtual cluster as a service, in: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 652–657. `doi:10.1109/CloudCom.2011.101`.
URL `http://dx.doi.org/10.1109/CloudCom.2011.101`

[13] IBM, IBM platform dynamic cluster, `http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/dynamiccluster.html`, [Online; accessed 10-December-2013].

[14] 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, special Issue on Cloud Computing. `doi:http://dx.doi.org/10.1016/j.jpdc.2010.05.005`.
URL `http://www.sciencedirect.com/science/article/pii/S0743731510000985`

[15] R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente, Elastic management of cluster-based services in the cloud, in: Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, ACDC '09, ACM, New York, NY, USA, 2009, pp. 19–24. `doi:10.1145/1555271.1555277`.
URL `http://doi.acm.org/10.1145/1555271.1555277`

[16] R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente, Multicloud deployment of computing clusters for loosely coupled mtc applications, IEEE Transactions on Parallel and Distributed Systems 22 (6) (2011) 924–930. `doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.186`.

[17] P. Marshall, K. Keahey, T. Freeman, Elastic site: Using clouds to elastically extend site resources, in: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, 2010, pp. 43–52. `doi:10.1109/CCGRID.2010.80`.

[18] P. Marshall, H. M. Tufo, K. Keahey, D. L. Bissoniere, M. Woitaszek, Architecting a large-scale elastic environment - recontextualization and adaptive cloud services for scientific computing., in: S. Hammoudi, M. van Sinderen, J. Cordeiro (Eds.), ICSOFT, SciTePress, 2012, pp. 409–418.
URL `http://dblp.uni-trier.de/db/conf/icsoft/icsoft2012.html#MarshallTKBW12`

[19] D. Duplyakin, P. Marshall, K. Keahey, H. Tufo, A. Alzabarah, Rebalancing in a multi-cloud environment, in: Proceedings of the 4th ACM Workshop on Scientific Cloud Computing, Science Cloud '13, ACM, New York, NY, USA, 2013, pp. 21–28. `doi:10.1145/2465848.2465854`.
URL `http://doi.acm.org/10.1145/2465848.2465854`

[20] M. D. 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, HPDC '09, ACM, New York, NY, USA, 2009, pp. 141–

150. doi:10.1145/1551609.1551635.
URL http://doi.acm.org/10.1145/1551609.1551635

[21] B. Javadi, R. K. Thulasiram, R. Buyya, Characterizing spot price dynamics in public cloud environments, Future Gener. Comput. Syst. 29 (4) (2013) 988–999. doi:10.1016/j.future.2012.06.012.
URL http://dx.doi.org/10.1016/j.future.2012.06.012

[22] S. Tang, J. Yuan, X.-Y. Li, Towards optimal bidding strategy for amazon ec2 cloud spot instance, in: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, CLOUD '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 91–98. doi:10.1109/CLOUD.2012.134.
URL http://dx.doi.org/10.1109/CLOUD.2012.134

[23] Y. Song, M. Zafer, K.-W. Lee, Optimal bidding in spot instance market, in: INFOCOM, 2012 Proceedings IEEE, 2012, pp. 190–198. doi:10.1109/INFCOM.2012.6195567.

[24] M. Mazzucco, M. Dumas, Achieving performance and availability guarantees with spot instances, in: High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, 2011, pp. 296–303. doi:10.1109/HPCC.2011.46.

[25] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, D. Tsafrir, Deconstructing amazon ec2 spot instance pricing, in: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 304–311. doi:10.1109/CloudCom.2011.48.
URL http://dx.doi.org/10.1109/CloudCom.2011.48

[26] M. Mattess, C. Vecchiola, R. Buyya, Managing peak loads by leasing cloud infrastructure services from a spot market, in: Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications, HPCC '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 180–188. doi:10.1109/HPCC.2010.77.
URL http://dx.doi.org/10.1109/HPCC.2010.77

[27] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, R. Buyya, The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds, Future Gener. Comput. Syst. 28 (6) (2012) 861–870. doi:10.1016/j.future.2011.07.005.
URL http://dx.doi.org/10.1016/j.future.2011.07.005

[28] Amazon, Amazon EC2 Spot Bid Price Limits, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-limits.html, [Online; accessed 2-December-2014].

[29] W. Voorsluys, S. K. Garg, R. Buyya, Provisioning spot market cloud resources to create cost-effective virtual clusters, in: Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I, ICA3PP'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 395–408.
URL http://dl.acm.org/citation.cfm?id=2075416.2075453

[30] M. Taifi, J. Y. Shi, A. Khreishah, Spotmpi: A framework for auction-based hpc computing using amazon spot instances, in: Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part II, ICA3PP'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 109–120.

URL http://dl.acm.org/citation.cfm?id=2075462.2075474

[31] MIT, Starcluster elastic load balancer, http://web.mit.edu/stardev/cluster/docs/0.92rc2/manual/load_balancer.html, [Online; accessed 9-December-2013].

[32] F. Alvarruiz, C. de Alfonso, M. Caballer, V. Hernández, An energy manager for high performance computer clusters, in: Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, ISPA '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 231–238. doi:10.1109/ISPA.2012.38.
URL http://dx.doi.org/10.1109/ISPA.2012.38

[33] C. de Alfonso, M. Caballer, F. Alvarruiz, G. Moltó, V. Hernández, Infrastructure deployment over the cloud, in: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 517–521. doi:10.1109/CloudCom.2011.77.

[34] J. V. Carrión, G. Moltó, C. De Alfonso, M. Caballer, V. Hernández, A Generic Catalog and Repository Service for Virtual Machine Images, in: 2nd International ICST Conference on Cloud Computing (CloudComp 2010), 2010.

[35] A. Works, Ansible software, http://www.ansibleworks.com/, [Online; accessed 4-December-2014].

[36] M. Caballer, I. Blanquer, G. Moltó, C. de Alfonso, Dynamic management of virtual infrastructures, Journal of Grid Computing 13 (1) (2015) 53–70. doi:10.1007/s10723-014-9296-5.
URL http://dx.doi.org/10.1007/s10723-014-9296-5

[37] M. A. Jette, A. B. Yoo, M. Grondona, Slurm: Simple linux utility for resource management, in: In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003, Springer-Verlag, 2002, pp. 44–60.

[38] AdaptiveComputing, Torque resource manager, http://www.adaptivecomputing.com/products/open-source/torque/, [Online; accessed 12-January-2015].

[39] J. Duell, The design and implementation of berkeley labs linux checkpoint/restart, Tech. rep., Lawrence Berkeley National Laboratory (2002).

[40] OpenVPN, Open Source VPN, http://openvpn.net/, [Online; accessed 11-December-2014].

[41] S. Yi, D. Kondo, A. Andrzejak, Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud, in: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 236–243. doi:10.1109/CLOUD.2010.35.
URL http://dx.doi.org/10.1109/CLOUD.2010.35

[42] D. Jung, S. Chin, K. Chung, H. Yu, J. Gil, An efficient checkpointing scheme using price history of spot instances in cloud computing environment, in: E. Altman, W. Shi (Eds.), Network and Parallel Computing, Vol. 6985 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 185–200. doi:10.1007/978-3-642-24403-2_16.
URL http://dx.doi.org/10.1007/978-3-642-24403-2_16

[43] S. Khatua, N. Mukherjee, A novel checkpointing scheme for amazon ec2 spot instances, in: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, 2013, pp. 180–181. doi:10.1109/CCGrid.2013.71.

[44] A. Pérez-García, F. Gómez-Martínez, A. Alonso, V. Hernández, J. M. Alonso, P. de la Fuente, P. Lozano, Architrave: Advanced analysis of building structures integrated in computer-aided design, in: Construction and Building Research, Springer Netherlands, 2014, pp. 123–130. `doi:10.1007/978-94-007-7790-3_17`.

[45] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. Epema, The Grid Workloads Archive, in: Future Gener. Comput. Syst. 24 (7) (2008) 672–686. `http://dx.doi.org/10.1016/j.future.2008.02.003`.

[46] NorduGrid dataset, The Grid Workloads Archive, `http://gwa.ewi.tudelft.nl/datasets/gwa-t-3-nordugrid/report/`, [Online; accessed 24-September-2015].