Received: 30 March 2018 | Revised: 15 August 2019 | Accepted: 12 December 2019
DOI: 10.1002/cpe.5662

WILEY

SPECIAL ISSUE PAPER

On automated cloud bursting and hybrid cloud setups using Apache Mesos

Hårek Haugerud | Noha Xue | Anis Yazidi

Department of Computer Science, Oslo Metropolitan University, Oslo, Norway

Correspondence
Anis Yazidi, Oslo Metropolitan University, PO Box 4, St Olavs plass, N-0130 Oslo, Norway.
Email: anis.yazidi@oslomet.no

Summary

Hybrid cloud technology is becoming increasingly popular as it merges private and public clouds to bring the best of two worlds together. However, due to the heterogeneous cloud installation, facilitating a hybrid cloud setup is not simple. Despite some commercial solutions being available to build a hybrid cloud, an open-source implementation is still unavailable. In this paper, we try to bridge the gap by providing an open-source implementation using the power of Apache Mesos. We build a hybrid cloud on top of multiple cloud platforms, private and public. Through comprehensive experimental results, we show that our solution is able to ensure resource bursting by leveraging the power of the public cloud.

KEYWORDS

Apache Mesos, cloud bursting, data segmentation, fault tolerance, open-source hybrid cloud

1 | INTRODUCTION

The use of cloud computing is becoming more common, bringing with it the advantages of flexibility and abundance of available resources, but also a greater degree of complexity as well as privacy and security concerns.

Nevertheless, cloud technologies are progressing and maturing each year, providing functionality for individual's personal use, as well as for enterprises with stringent requirements on performance, availability, and price. A possible solution is to utilize multiple cloud providers to minimize the risks of service disruption and degradation. In addition, a platform utilizing multiple clouds will enable organizations to pick one or several cloud providers based on various factors such as price, location, availability, and performance among many others. Another viable option would be to use private data centers in addition to external cloud providers in a hybrid setup. This type of setup, which is often referred to as hybrid cloud, is becoming increasingly popular as more companies are starting to invest in and offer these solutions. According to Google Trend Search, there has been increasing interest in the term, hybrid cloud, since the beginning of 2009.[1] However, most solutions in the market are either proprietary or not open-sourced, which is not ideal if the purpose of using multiple cloud providers is to avoid vendor lock-in.

Even with the possibility of using multiple cloud providers and private data centers, there is still the issue of static partitioning and isolation of resources due to the design of virtualization. Static partitioning of systems makes it difficult to fully utilize the resources due to fluctuations in system use, which may be affected by factors such as business hours, holidays, and batch processing, to name just a few.

This paper will explore and document the attempts to design and prototype a possible open-source solution for constructing a computer cluster built on top of private servers and external cloud providers. Moreover, investigations will be carried out on utilizing spot instances for cloud bursting purposes and for segmenting applications based on certain parameters, as well as high availability solutions, leveraging multiple clouds as one possible way of further minimizing downtime.

In this paper, we propose an open-source solution to building a computer cluster on top of private computer resources in addition to public cloud providers. By utilizing both private hardware and public cloud providers, improved levels of availability through different failure scenarios are achieved. We demonstrate how it is possible to segment data and process it to specified locations or groups. Segmentation denotes the possibility of dividing the cluster into logical segments based on certain variables such as location, performance, and any other desired factor. The segments should be isolated from each other. Furthermore, it also refers to the ability of the solution to restrict tasks to be run on those divided segments. Some preliminary results regarding those aspects have appeared in a previous study.[2] In this paper, we provide more details about the design of our solution. Most importantly, we focus on the bursting aspect and demonstrate that our solution is able to leverage spot price instances to accommodate cloud bursting.

The remainder of this paper is organized as follows. In Section 1, we introduce and motivate the problem. Section 2 presents a comprehensive survey of the state-of-the-art work on hybrid clouds together with a brief introduction to the basic concepts of Apache Mesos. Section 3 provides insights into the design of our solution, while Section 4 gives details of the implementation of our solution for building a hybrid cloud based on Apache Mesos. Furthermore, we provide some experimental results that demonstrate the three aforementioned properties that characterize our solution, that is, resource bursting, data segmentation, and fault tolerance. Finally, we discuss the results in Section 5 and draw conclusions in Section 6.

2 | BACKGROUND AND RELATED WORK

Computer clustering can loosely be defined as a group of hardware connected together to provide a single virtual and powerful hardware platform. With the added layer of abstraction clustering gives, it is possible to partition the hardware dynamically using software. Services running on top of a cluster can therefore dynamically scale and move within the cluster without being limited by the underlying hardware partitioning. Large companies use this type of flexible partitioning for their services, for example Google with its self-developed platform Omega[3,4] and Twitter with Apache Mesos.[5]

2.1 | Apache Mesos

Apache Mesos is a distributed system kernel that abstracts hardware resources like CPU, memory, and storage to construct a dynamically partitioned computer cluster. Due to similarities in how an operating system abstracts hardware, Apache Mesos has been referred to as a "datacenter OS."[6-8] Twitter has been one of the main driving forces behind the development of Apache Mesos,[9,10] and it uses it for various core services in production.

Apache Mesos provides a uniform computer environment by adding an abstraction layer between the hardware and software frameworks and handling resource allocation between those parts as shown in Figure 1.

Various parts of Apache Mesos work to provide a functioning and robust master-slave setup, as well as scheduling and executing tasks given by a framework in the distributed environment. Apache ZooKeeper is the subsystem responsible for coordinating the election of master nodes and managing fail-over should a master node fail to respond. Apache Mesos operates with a layered resource negotiation handled by an allocation module located at the master node. A framework-specific scheduler will receive and process resource offerings, while a framework specific executor running at the slaves will allocate resources and launch the tasks given. The general setup is shown in Figure 2.

Apache Mesos provides isolation for the running tasks using Linux containerization with cgroups. It is the Mesos frameworks that provides utility to the cluster, also referred to as *Mesos applications*. Through the schedulers, the frameworks receive resource offers and submit tasks

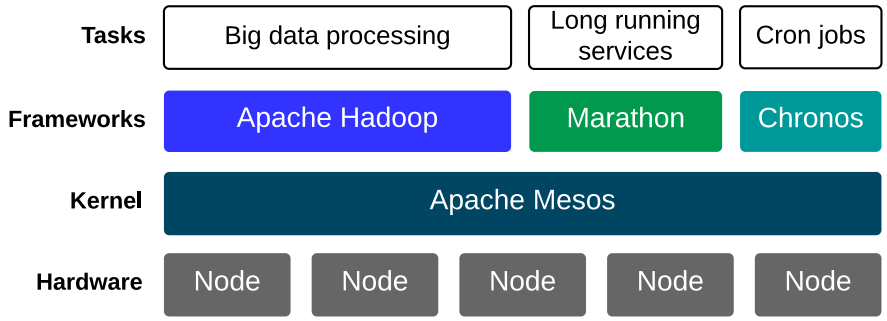

FIGURE 1 Abstraction model of Apache Mesos and some related frameworks

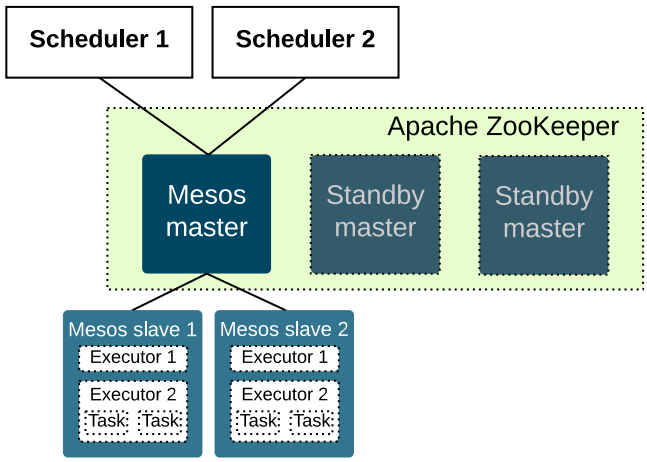

FIGURE 2 A simplified model of how tasks are scheduled and executed

to be launched. Marathon on top of Apache Mesos provides a robust platform for running long-term services, making it easier to achieve high service-level agreement (SLA) and to scale services. While Apache Mesos manages hardware redundancy, Marathon does the same for applications. The Marathon framework will ensure that the specified applications are running as long as Marathon is running, while Apache Mesos ensures that Marathon is running as long as there is a bare minimum of nodes running and idle resources are available. The Hadoop framework can be run on top of Apache Mesos. Compared to a standalone Hadoop cluster, Apache Mesos provides an easier and more flexible way of managing the cluster itself. The Chronos framework was created as a distributed version of Cron, and it schedules and runs jobs at specified intervals.

Yet Another Resource Negotiator (YARN),[11] was developed as a resource manager and scheduler for the next generation of Hadoop. YARN can be looked upon as a competitor of Apache Mesos, as both technologies attempt to solve the same problem, albeit with different strategies. Apache Mesos utilizes a two-level resource scheduling strategy and was developed as a general type of scheduler, while YARN opts for a monolithic approach mainly focusing on Hadoop. However, despite the technologies being similar and arguably competing, an Apache Mesos framework that utilizes YARN has been developed called Apache Myriad.[12]

2.2 | Cloud bursting

A specific workload deployment model called cloud bursting utilizes a hybrid cloud solution to load balance a workload between private computer resources and public clouds.[13,14] In this model, workloads are mainly processed in-house using private resources, with the possibility to "burst" out into public cloud providers should the workload be too much for the in-house resources to handle. As shown in Figure 3, this allows an organization to dimension their data center for average workloads and deal with the spikes using the public cloud and only pay for the extra computer resources when used.

2.3 | Related work on hybrid clouds

The concepts of multicloud and hybrid cloud are not new and several companies explore and capitalize on these concepts. Most of the available solutions are commercial. Different vendors including Dell, IBM, and HP provide hybrid cloud solutions.[15,16]

The MODAClouds project[17] utilizes several tools to provide an environment for utilizing multiple cloud providers. Several large companies offer hybrid cloud solutions, often in conjunction with their existing product portfolio. VMWare offers a hybrid cloud solution called *vRealize suite*, which provides one interface to manage the entire hybrid cloud platform.[18,19] Other companies such as Cisco,[20] IBM,[21] and RackSpace[22] also offer hybrid cloud solutions. An IBM hybrid cloud solution is an example of another commercial solution that addresses the challenges of managing heterogeneous virtual environments to create a hybrid cloud platform.[16] PaaSage is an interesting initiative to build a hybrid cloud solution using a defined deployment model, Cloud Application Modeling and Execution Language (CAMEL).[23]PaaSage applications specify tasks according to the CAMEL model, which will then be processed and deployed on multiple clouds according to the specified requirements. However, the dependency of PaaSage on the CAMEL language presents a usability barrier that might limit its popularity as system administrators need to learn yet another language. Multicloud Deployment of Computing Clusters for Loosely Coupled MTC Applications[24] explores the concept of deploying a computer cluster on top of a multicloud environment. However, there are some aspects of the paper that do not fully translate to the practical issues a system administrator might encounter. For instance, the clustering technology used in this study was Sun Grid Engine (SGE), which is a long time veteran of approximately 15 years. Incidentally, Apache Mesos was designed to address some of the design weaknesses of SGE,[25] in particular the use of static partitioning for the jobs run in the cluster, which prevents fully efficient utilization of the resources.[25]

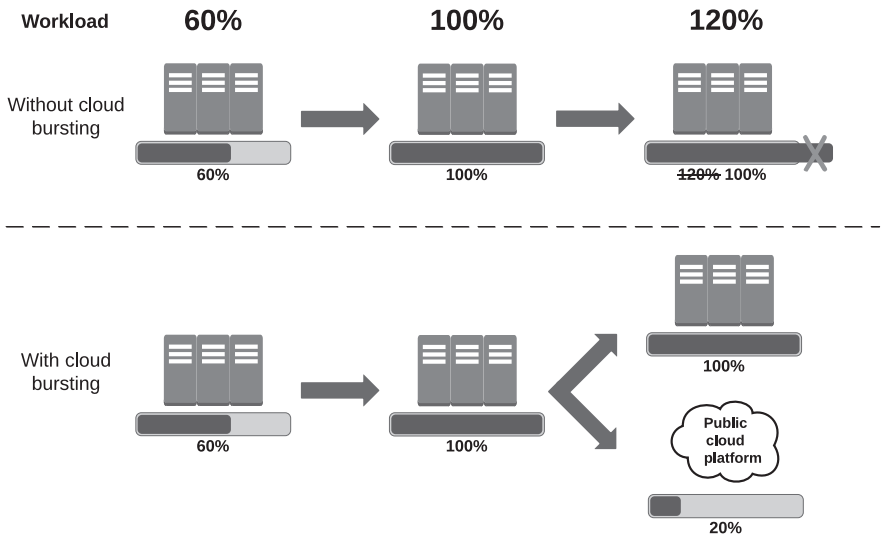

FIGURE 3 Illustration of how cloud bursting works when the capacity limit of the private data center is exceeded

However, there has been no practical demonstration of open-source and freely available clustering technology being used to attempt to address the multitude of challenges involved in creating a hybrid cloud platform that is available and supports data segmentation. This paper outlines an attempt to prototype such a solution in addition to facilitating cloud bursting using spot price instances.

Bittencourt et al[26] treated an underinvestigated aspect of scheduling workflows in hybrid cloud: the communication link between the public and private cloud that could become a bottleneck in cases of communication-intensive workflows. Therefore, the authors propose different communication-aware scheduling algorithms that take into account the fluctuations of the link load between the bridged clouds.

Borja et al introduced OpenNebula,[27] which is one of the most popular open-source virtual infrastructure managers. OpenNebula permits the abstraction of resources from an existing local grid and a cloud infrastructure. However, installation and management of OpenNebula is fairly complex as it consists of many parts to facilitate the rich feature set of a virtual machine (VM) manager. With virtualization of infrastructure, resources, network, and storage comes many vectors for failure, and SLA rates may also be affected. Additionally, with clustering technologies like Apache Mesos, VMs may not be necessary and may even degrade performance due to virtualization overhead.

At the heart of OpenNebula, we find Haizea,[28] which is a VM-based lease manager that enhances the resource scheduling manager by including advanced reservation of resources and queuing of best effort requests. Guo et al[29] extended OpenNebula to support cloud bursting across public and private clouds[29] by implementing a cloud management layer exposed through an XML Remote Procedure Call API. The API exposes the needed functionalities for cloud bursting including mainly: resource monitoring, VM managing, VM migration, and VM precopying. However, the bursting in the latter solution is achieved through either migrating VMs across public and private clouds, or using the so-called Precopier. The Precopier replicates selected VMs at the filesystem level using rsync. In contrast to our solution, the bursting mechanism does not require VM migration or precopying. In fact, the bursting takes place at the task level and is handled in a transparent manner by Mesos. Furthermore, OpenNebula suffers from the single point of failure problem.[30] In this paper, we present a lightweight solution, which is failure resilient. Similarly, it is also possible to create hybrid cloud using AWS and Eucalyptus.

While the above approaches focus on abstracting resources of heterogeneous cloud infrastructures as well as grid infrastructures, another body of approaches focuses on VM migration between heterogeneous clouds with different virtualization technologies. It is known that different infrastructures and hypervisors are an impediment to free migration of VMs between different clouds.

Kangaroo[31] was designed for migrating VMs between geographically distributed OpenStack clouds. The main idea behind the approach is to create a virtual networking switch for communication between nested VMs over different infrastructure clouds.

Xen-Blanket[32] bridges different clouds and allows migration between KVM and Xen. Xen-Blanket builds a second layer on top of Xen to homogenize various cloud platforms.

Bicer et al[33] proposed middleware for facilitating cloud bursting for a type of MapReduce processing in a geographically distributed data set. However, the latter work focuses mainly on deploying MapReduce over the cloud where the data could be split between a local cluster and a cloud resource. The study also opens for the possibility of work stealing, which means that data at one end is processed using resources from the other end. The middleware is able to limit remote data retrieval overheads. In another research paper[34] the authors extended their framework and achieve a compromise between costs and deadlines. Other studies rather focus on the advantages of building hybrid cloud software using open-source tools.[35]

It is worth mentioning that a body of research[36-38] focuses on simulating different strategies for leveraging the power of hybrid cloud to achieve bursting at a lower cost by dividing the load between private and public clouds. In one of these approaches, various algorithms were used to reduce the execution cost by leveraging price difference between different clouds.[37] The authors devised an algorithm that decides bid prices, when to use local machines, and when to checkpoint and migrate the job between these resources, with the goal of completing the job on time and at the minimum cost.

In another work,[39] six different bursting strategies were investigated to achieve a compromise between cost and performance. However, the evaluation was done using a discrete-event simulator. In the work reported by Peláez et al,[40] different algorithms have been proposed for scheduling deadline-constrained bag-of-task workloads in a hybrid cloud infrastructure. Those scheduling algorithms use online processing time estimators and aim to reduce the infrastructure cost while meeting the deadlines. The authors resorted to the Cloudsim toolkit, which is a cloud simulator to simulate the scheduling in two public clouds, namely Amazon EC2 and Google Compute Engine. Those sophisticated algorithms are useful in the context of this current study to further optimize the execution cost in the case of bag-of-task workloads. The Reservoir project[36] is a multicloud project aimed at federating two or more independent cloud providers. However, standardization is necessary before federation can be rolled out in realistic-cloud settings that go beyond merely being a testbed. The different spectrum of standards governing different cloud solutions is a barrier to transparent use of multicloud platforms.

3 | DESIGN

This section will describe the proposed solutions, discovered limitations, and additional considerations regarding the setup of an Apache Mesos cluster in a hybrid cloud setup. In addition, a solution for automating cloud bursting will be proposed that leverage spot price instances to maximize the cost effectiveness of the proposed solution.

TABLE 1 The specifications for the instance types chosen for running Mesos master nodes

	Altocloud	AWS EC2
	m1.medium	**m3.medium**
vCPUs:	2	1
RAM:	4096 MB	3.75 GiB

The proposed designs and solutions in this section will be tailored to the testbed environment. Further details, adjustments, and specifications regarding the testbed will described in this section.

3.1 | Environment

As alluded to, the choice of instance type for the Mesos master node has to be given extra consideration, as it is the only part of the cluster that results in almost fully idle resources. In one approach,[8] a series of performance benchmarks was conducted to gauge the capacity of certain parts of the Apache Mesos cluster. Using an 8 vCPU and 6 GB RAM instance at Amazon Web Services EC2 as the Mesos master node, the scheduling of tasks and internal processing required for the cluster adds an overhead of less than one second, with 50 000 Mesos slave nodes in the cluster. The paper was published in 2011, and it is assumed that further performance optimizations have been done since then, reducing the overhead even further.

For this work, a medium instance type will be used for the Mesos master nodes. For Altocloud this would be *m1.medium*, while at EC2 it would be *m3.medium*. While a small instance type might have worked out, a medium instance type was chosen. This was done as a safety measure, as too low performance might result in unexpected glitches, especially regarding network performance. To be sure that this would not be an issue, the instance type was over-dimensioned by a considerable margin. The focus on evaluating and designing the cluster for high availability, segmentation of the data in particular, and cloud bursting scenarios. Performance as a factor is less of a priority. The specifications are for the medium instance types have been relisted for convenience in Table 1. It should be noted that it is common to run Mesos frameworks on the same machines as the Mesos master nodes, and the machines should be dimensioned for this extra load, unless the frameworks are running on separate machines.

Since it was determined that Amazon Web Services would be the public cloud provider for the hybrid cloud setup, the availability region has to be determined. When deciding the availability region to be used, it is important to consider price, latency, and regulations, among other things. Due to the timeframe for the project, the overall price difference will be small. The latency of the availability region was therefore given priority. As a result, the availability regions EU Central (Frankfurt) was picked as the primary choice, with EU West (Ireland) as the secondary choice.

3.2 | Architecture

This section will contain the proposed solutions in three main parts on availability, segmentation of data, and automated cloud bursting.

3.2.1 | Availability

Depending on the required level of availability needed, it is necessary to consider fault tolerance at multiple levels, from the fundamental level of hardware resources up to the individual applications. A key technique for improving availability is to duplicate and keep a redundant copy or backup of the entity one wishes to improve. This could mean physical machines, network links, power circuits, or multiple instances of an application. In most cases, it is also possible to use the redundancy to load balance, which lowers the overhead of keeping a duplicate.

However, it is imperative that the redundant entities are as independent of each other as possible in order minimize the risk of failure. Consider the following scenario: A service is running on two servers located at a single data center and load balancing distributes the load between them. While the servers are separate, they both depend on the data center being functional, although it can experience full blackout due a disaster of some sort. To improve availability, the servers should ideally be placed in two separate data centers.

To achieve higher levels of availability, Apache Mesos can be deployed on multiple cloud platforms and data centers, forming a hybrid cloud. Several things have to be considered before setting up an Apache Mesos cluster. In an Apache Mesos cluster, the Mesos master node is responsible for managing the cluster and is therefore vital to the cluster. The number of master nodes in a cluster must be an odd number. This is in order to prevent a split brain problem, which occurs when the cluster splits into two or more smaller clusters, each with its own Master node managing the cluster. This is a problem as it results in inconsistent states for the cluster as a whole and must be prevented. With an odd number of Mesos master nodes correctly configured, more than half of the Mesos master nodes need to be able to communicate to change the state of the cluster.

Prototype 1: Maximizing availability

Figure 4 illustrates a solution using three separate cloud platforms as components of the hybrid cloud. The Mesos master nodes are distributed between the availability zones, with a few Mesos slave nodes arbitrarily present in each of the zones. The failure rate of each of these zones is assumed to be highly independent of the others, which will lower the risk of failure of the underlying resources used by the Apache Mesos cluster. For this prototype, the officially recommended number of three Mesos master nodes has been chosen. This will suffice for a proof of concept. Should the underlying resources the Mesos master nodes utilize have a high failure rate, a higher number of Mesos master nodes should be considered. This setup will maximize availability, which is ideal for public facing services that do not have any particular requirement other than

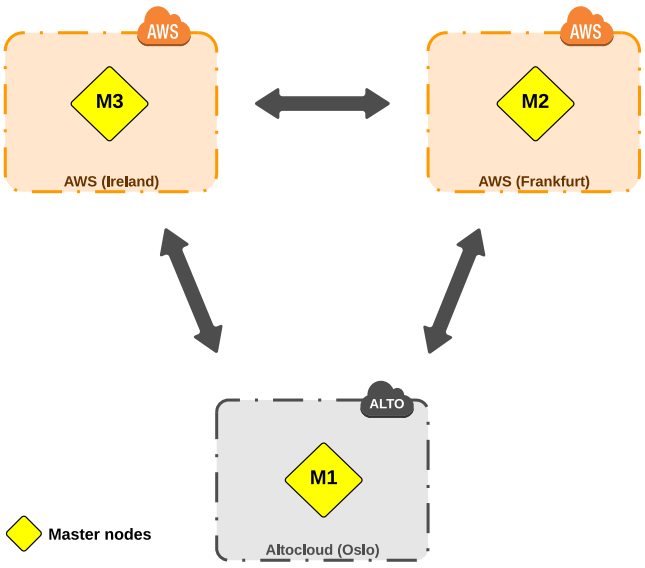

FIGURE 4 A hybrid cloud setup, distributing the Mesos master nodes to independent availability zones

to maximize availability. Public services such as a website, back-end infrastructure for advertising services, or a smartphone application are examples of this type of services.

Due to the architecture of Apache Mesos, any slave node can become unavailable without affecting the overall service availability. As long as the framework deploying the services is fault-tolerant and there are enough slave nodes to be able to accommodate the bare minimum of processing needed to keep the services running, a large amount of slaves can fail simultaneously. Moreover, this setup allows for downtime in an entire availability zone, and the hybrid cloud platform will still be functional in such a scenario.

This setup requires each and every node of the network to have a public IP-address that is routable on the Internet. This is a huge limitation, as the number of IPv4 addresses is limited. Alternatively, the number of Mesos slave nodes deployed on Altocloud can be reduced or removed altogether, relying on the public routable IP-addresses Amazon Web Services EC2 assigns to each node by default. Another solution is to set up VPN tunnels between the availability zones and to route using private addresses.

Prototype 2: Prioritize local availability

This prototype is illustrated in Figure 5. In this suggested setup, local availability is given priority, with the majority of the Mesos master nodes present in the local cloud. This particular setup uses five master nodes, but it can easily be extended to include a higher number as long as the majority of the Mesos master nodes are located in the local cloud.

Even if the access to the Internet is lost, local access can be obtained from the same network as the local cloud, and the services running on the cloud will continue to be responsive for internal use. This makes this type of setup ideal for local services like Enterprise Resource Planning (ERP) applications, local batch jobs, and analysis frameworks like Hadoop running, which is not normally publicly accessible. The proposed setup

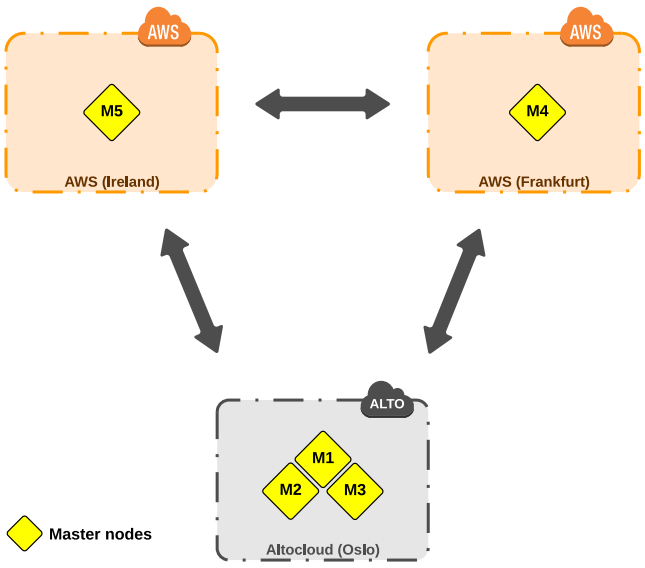

FIGURE 5 A five Mesos master node cluster with the majority of them located at Altocloud

will function well where the private cloud or data center is used as the preferred baseline location, with public cloud resources added to increase processing power and/or availability.

Fallback solution: VPN tunneling

A fallback solution has been prepared in order to take account of the possibility that Apache Mesos does not work as intended for the proposed prototypes or any other networking issue that may cause a problem. In such cases, an instance dedicated to function as a VPN gateway will be set up at each site, which will establish VPN tunnels between them. This can be done due to the amount of control Amazon Web Services give over the network through their VPC service. Likewise, Altocloud, being an OpenStack installation, also permits a higher level of manipulation of the network. IPSECv2 tunnels will be set up at VPN gateways and installed and configured through Openswan, an IPSEC implementation that supports an array of features, with network address translation (NAT) traversal being particularly interesting. As mentioned previously, Apache Mesos is currently developed and maintained to work in private network configurations and, given the way public IPs are handled by the cloud platforms, NAT traversal may end up being the main issue. Correctly set up, a VPN environment will function just like any other private network for the machines present in the network.

3.2.2 | Segmentation of data

The segmentation of the hybrid cloud can be facilitated using existing functionality in Apache Mesos and the Marathon framework. Every node in an Apache Mesos cluster can be tagged with an arbitrary number of attributes, which will be sent together with the resource offers from a Mesos slave node. By reading the attributes in combination with the offered resources, the frameworks can make a decision on whether or not to accept the offered resources. In addition to Marathon, there are other Mesos frameworks that manage long-running services such as Aurora and Singularity. However, Marathon was picked because of its simple graphical interface, well-documented REST API, and a clear-cut way of enforcing constraints, which is vital for segmenting the hybrid cloud.

Marathon supports a number of operators setting constraints on how applications are run, with *CLUSTER* being the operator of interest. This operator will require the tasks to run in a cluster constrained by the defined attribute. To constrain an application to only run on nodes where the attribute *color* is set to *blue* then a constraint will look like this: *color:CLUSTER:blue*.

To segment the data, an attribute will be set on every slave node in the Mesos cluster. The attribute will be named *cloud_type* and will contain either *public* or *private*. Attributes can be set when starting the Mesos services or preconfigured in files. Another way to segment the cluster is to create an attribute named *clearance_level* and have values that mimic the clearance levels of public, confidential, secret, and top secret.

In Section 4.3, a practical example of how this security precaution can be applied is demonstrated. The tasks run are tagged with an attribute named `cloud_type`, which is set to `private`. This ensures that the task may only run on the local private cloud and will never be migrated to any of the public clouds.

In terms of use cases where privacy is the priority, Apache Mesos can be deployed only at compliant sites. Compliant sites means in this case both private locations and certified cloud providers. For instance, Amazon cloud provider has an availability region named GovCloud, which satisfies several US requirements related to privacy.

3.3 | Automated cloud bursting

A cloud bursting solution with a hybrid cloud setup utilizing Apache Mesos to provide an abstraction layer on top of physical or virtual resources makes cloud bursting as simple as adding a Mesos slave node located in the cloud. To bootstrap the slave nodes, a simple bash script is supplied as user data at instance launch. The script will install and configure the necessary services to make it register itself to the cluster. This adds an approximate of three to four minutes of overhead when deploying a Mesos slave node. This overhead would most likely not be improved with the use of any configuration management tool, on the contrary, it would most likely add additional overhead. To shorten this time, an image can be prepared beforehand with the necessary services configured.

The challenge is therefore to automate it and to utilize the spot price instances, while considering the billing-cycle interval of one hour and price fluctuations. A prototype will be written in Python that will run on top of Marathon in a fault-tolerant manner.

The script will consist of three main parts:

- Data collection
- Price and scaling decision logic
- Management of the spot requests and instances

Data collection is the first part of the script. It encompasses the collection of metrics from Apache Mesos and through the Amazon Web Services API that will help in the decision-making process. Interesting metrics include total resources available, resources in use, spot requests pending, number of spot instances, and metadata connected to each of the spot instances. Using the collected information, a decision is made on whether to scale the cloud bursting capacity up or down. Depending on how many factors and considerations one wishes to take into account, this step is potentially highly complex.

A related work[41] describes five bidding strategies that can be used. The bidding strategies are listed up in Table 2, where $G = 0.001$, which is the lowest granularity value permitted to be used at Amazon Web Services when bidding for spot instances.

As there are several aspects of the proposed algorithms in the mentioned paper that do not apply directly to the hybrid cloud scenario, a simpler decision algorithm has been devised. In Figure 6, the projected activity flow is illustrated.

On the launch of the script, the values of available resources at the private location will be provided through a configuration file. Based on the total available baseline resources and the resources currently in use, a percentage is calculated. This percentage will be the sole variable that will determine whether or not to utilize public cloud resources. In a live production, a more complex decision algorithm should be considered, since the one proposed for this prototype does not take account of any edge cases.

The script will operate with a *burst point threshold*, which represents the usage-percentage of the available resources at which the cloud should scale up. For instance, if the *burst point threshold* is set to 0.70, the script will request spot instances at when resource usage reaches 70%. Additionally, a specific maximum limit will be artificially set in order to prevent the prototype from scaling up too much and incurring huge costs.

The decision to scale down will be based on actual usage of the resources and the lifetime of a spot instance. Lifetime has to be considered, due to the hourly billing cycle that applies when the termination of an instance is initiated by the user. To avoid wasting already charged resources, a spot instance will only be terminated if the lifetime of the instance is nearing the next hourly billing cycle. For example, a spot instance may only be needed for 20 minutes, but will not be user-terminated for at least 30-35 more minutes to capitalize on the already charged hour.

TABLE 2 List of five possible bidding strategies for the spot price instances at Amazon Web Services EC2[41]

Bidding strategy	Bid value definition
Minimum	The minimum value observed in the price history + G
Mean	The mean of all values in the price history
On-demand	The listed on demand price
High	A value much greater than any price observed
Current	The current spot price + G

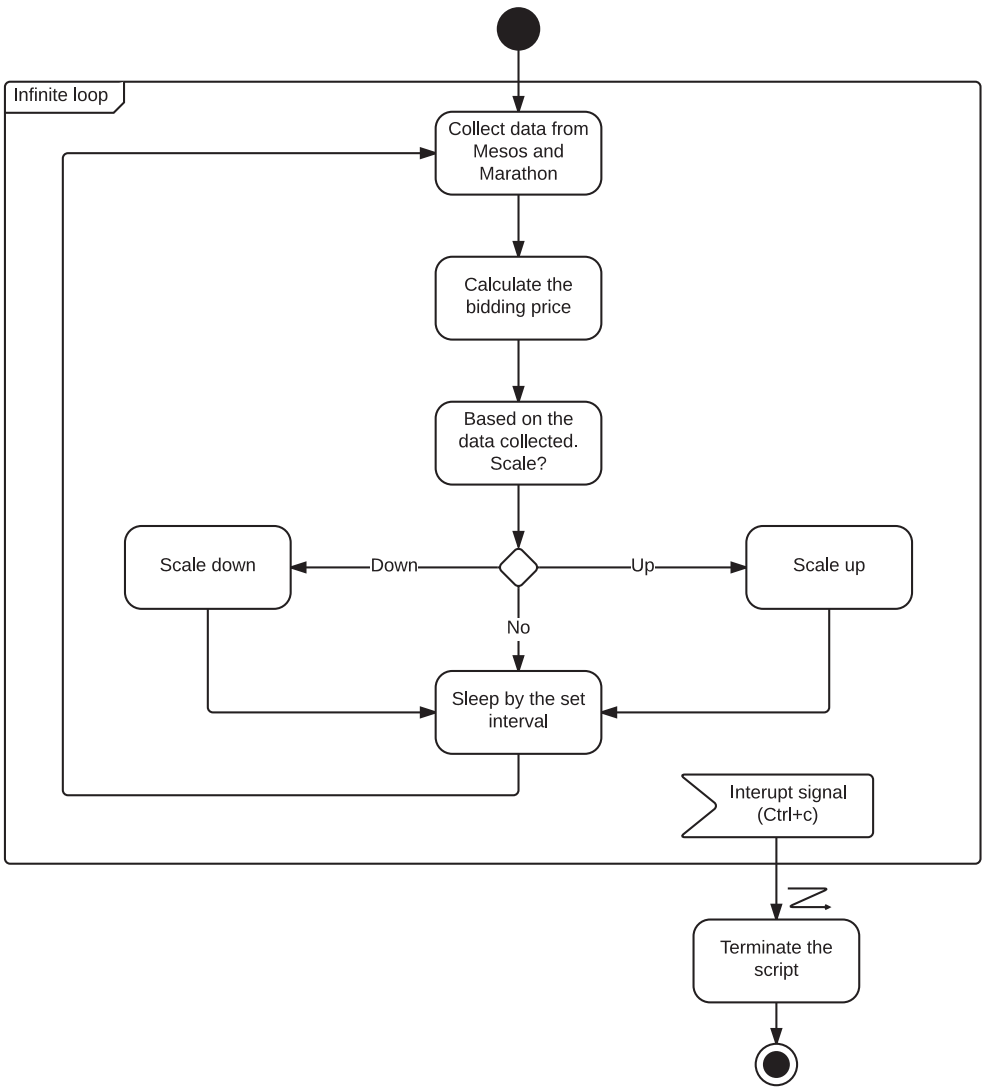

FIGURE 6 Projected activity flow for the script illustrated in an activity diagram

The chosen bidding strategy is the *current price + x*, where *x* is a predefined variable that is supplied with the script at launch time. *x* serves as price padding to avoid bidding at the exact market price, since bids at that level will be highly contested. By setting *x* to a high value, it is possible to increase the likelihood of the instance lasting longer before being terminated. On the other hand, by setting a low value, the instances could potentially be terminated due to increased prices, which would result in the charge for the last partial hour being waived. In the end, the script will bid the lowest price possible plus a padding value, which can be arbitrarily set to tweak the bidding strategy.

4 | IMPLEMENTATION

This section describes the actual implementations of the proposed prototypes and solutions. Our implementation is open source and available to the general public.[42]

4.1 | Setting up the testbed and Apache Mesos

The testbed uses Altocloud and Amazon Web Services VPC to emulate a private site and public cloud platforms, with Apache Mesos as the chosen clustering technology.

There are several ways of installing and managing Apache Mesos. Configuration management tools can be used to bootstrap and manage the Apache Mesos binaries in addition to other miscellaneous configuration. However, to have full control of the installation process, Apache Mesos was installed manually. We opted for manual installation because the deployment prototypes are nontraditional and of an experimental nature. In contrast to modifying and tweaking existing configuration management templates, manual installation gives more control of the installation process, easing the debugging process. When designing a solution for a production environment, clearly, it is desired to use a deployment tool like Ansible to easily reproduce the experimental settings.

Several issues were encountered during the installation of Apache Mesos master nodes. One of the issues caused the Mesos master nodes to not reach full equilibrium, resulting in a new leader being elected every minute, flip-flopping between the master nodes present. This was found to be a DNS-related issue. It was mitigated through an entry in the `/etc/hosts` file, which effectively functions as a manually maintained and makeshift DNS entry. In addition, due to a bug in ZooKeeper, an Apache Mesos process will crash if proper DNS handling is not in place.[43,44]

An Apache Mesos cluster that included both master nodes and slave nodes was successfully installed and configured in Altocloud, with slave nodes correctly registering themselves to the cluster through the leading master node. However, when attempting to register a slave node running at Amazon Web Services EC2, some peculiar activity was observed. The slave node located at EC2 managed to successfully send a registration request to the leading master node, passing through multiple layers of network abstraction, including two layers of NAT. Although the master node receives the registration requests, no registration acknowledgement is ever sent back.

Eventually, the cause was discovered to be a combination of the use of NAT and the way Mesos nodes communicate with each other. When a slave node sends a registration request, it includes information about the resources available and an IP address. The IP address is the one that is defined on the network interface bound by the Apache Mesos process. Furthermore, in a cloud environment like Altocloud and Amazon Web Services EC2, the public IP addresses are loosely coupled with the VM and function in a similar way to NAT. Consequently, the Mesos master attempts to send the acknowledgement and other internal traffic meant for that slave node to the nonroutable private IP address. The communication flow is illustrated in Figure 7.

As previously indicated in Section 3, NAT traversal appears to be an issue for Apache Mesos. To mitigate this limitation, a VPN solution has been deployed as outlined in the previous section. As the setup of a VPN solution is outside the scope of this paper, it will not be explained in detail. The subnets of the VPN network were divided into three main parts for the sake of simplicity, using all three private address spaces as defined by RFC1918. The resulting network is listed in Table 3.

A good chunk of the `172.16.0.0/12-subnet` has been left unallocated in case a need arises for additional private addresses.

4.2 | Availability

4.2.1 | Prototype 1: Maximizing availability

Using VPN tunneling, there is no need to allocate public IP addresses to each node for the purpose of maintaining the cluster, since the private IP addresses are routable within the hybrid cloud platform. With the exception of the extra infrastructure needed to maintain a VPN, the prototype is identical to the proposed design. Figure 8 illustrates the final implementation of the prototype, showing how the Mesos master nodes are distributed between the different availability regions.

Table 4 shows the subset of available slaves that is listed up in the graphical user interface (GUI) of Apache Mesos. Note the subnet differences of the slaves, which indicates where the slaves are located.

As for the Mesos master nodes, they were installed manually on top of instances booted up in their respective availability zones. The specifications for the Mesos master nodes are listed in Table 5.

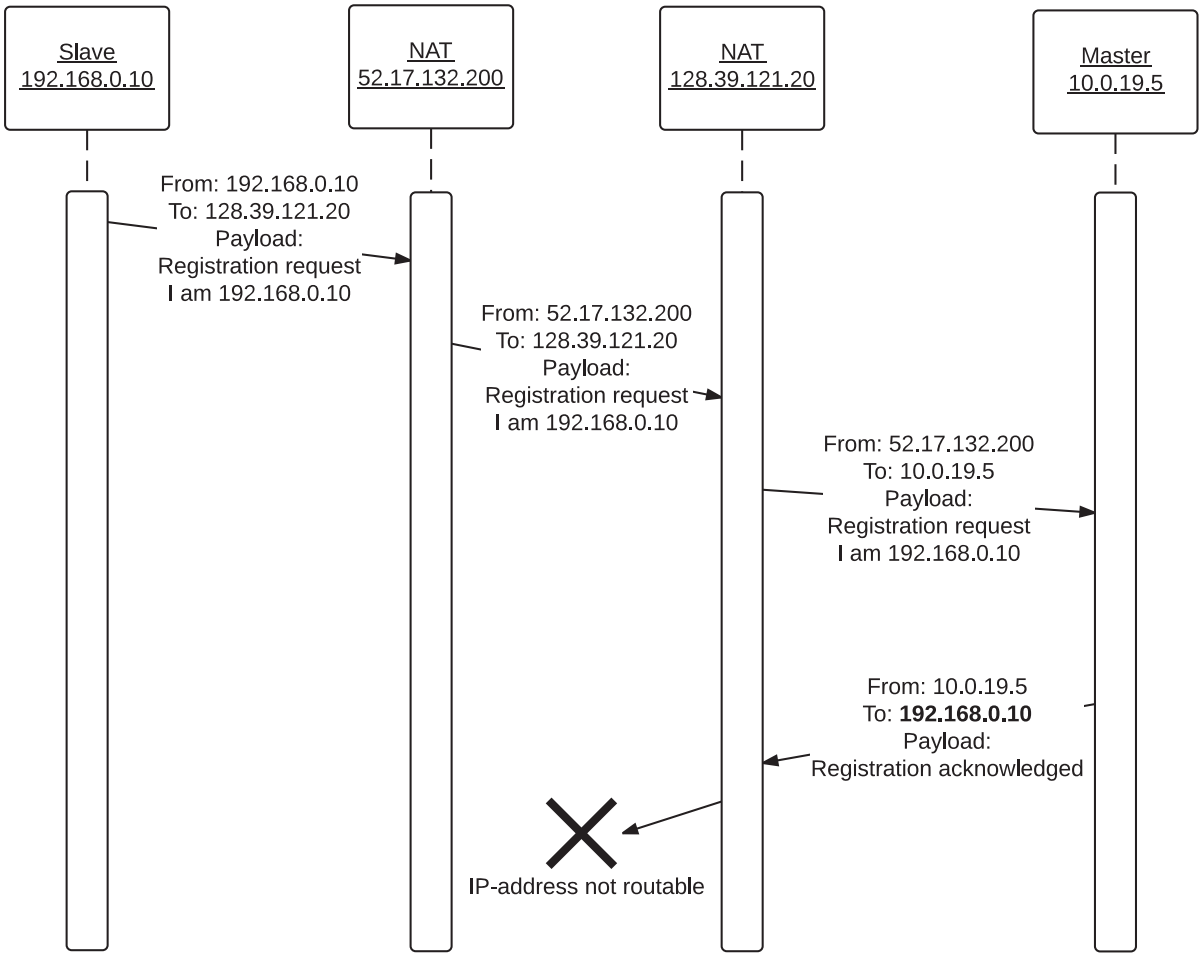

FIGURE 7 Communication flow between an Apache Mesos slave node and master node with the registration attempt failing due to how public IP addresses are handled in cloud platforms

TABLE 3 The network partitioning of the RFC1918 private addresses divided into separate subnets

Availability region/Site	Subnet
Altocloud (Oslo)	10.0.19.0/24
AWS (Ireland)	172.16.0.0/16
AWS (Frankfurt)	192.168.0.0/16

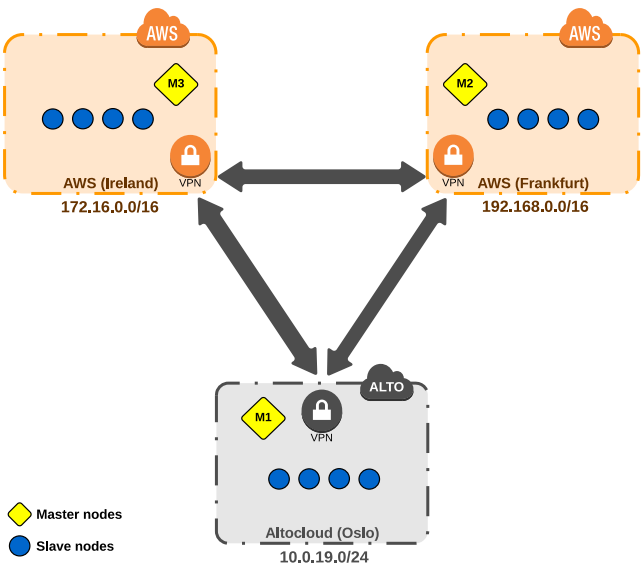

FIGURE 8 Prototype 1: Maximizing availability. Distributing the master nodes and thereby the risks

Slave ID	Host	CPUs	Mem	Disk
5050-5669-S0	192.168.187.205	1	496 MB	3.9 GB
5050-5669-S2	192.168.178.239	1	2.7 GB	3.9 GB
5050-5669-S1	172.16.231.155	1	496 MB	3.9 GB
5050-900-S71	10.0.19.9	4	6.8 GB	73.7 GB
5050-900-S69	10.0.19.8	4	6.8 GB	73.7 GB

TABLE 4 A subset of available slave nodes in this scenario. The information is taken from the Apache Mesos GUI and represents a truncated view of the available slaves

Master ID	IP-address	Location	Instance type
1	10.0.19.5	Altocloud (Oslo)	m1.medium
2	192.168.0.5	AWS (Frankfurt)	m3.medium
3	172.16.0.5	AWS (Ireland)	m3.medium

TABLE 5 A list of specifications for the Mesos Master nodes

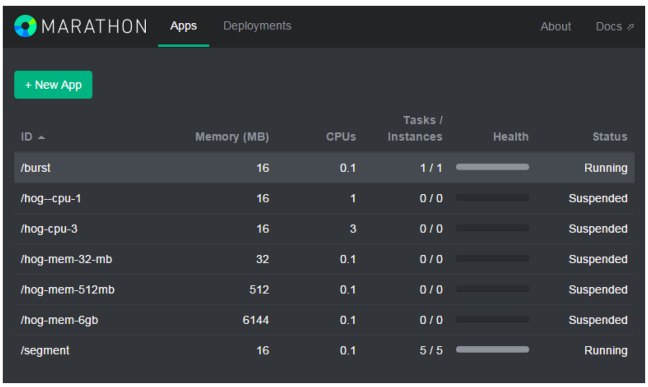

FIGURE 9 A screenshot of the Marathon GUI with some created tasks available to scale up or down

The Mesos slave nodes are set up using a bash script that is supplied as user-data at instance start-up. The meta framework Marathon has been installed to create tasks for testing availability. Marathon exposes a small subset of the functionality of the Marathon REST API, through a very simple GUI. Although simple, it is sufficient for creating, destroying and scaling tasks. A screenshot of the GUI is shown in Figure 9.

A Mesos slave process becomes unavailable

In the event that a Mesos slave node becomes unavailable for some reason, the Mesos master node allows a default timeout period of 75 seconds to pass before procedures for deactivating the slave node are begun. Should the slave node start responding within this timeout period, nothing will happen and both the Mesos master node and the slave node will simply ignore the temporary unavailability. However, if the timeout period is exceeded and the slave nodes are still unavailable, the Mesos master node will attempt to deactivate the Mesos slave process on the slave node preceding it on the list of available slave nodes. Tasks that were lost will be rescheduled to other slave nodes with available capacity. In Listing 1, an excerpt of the Mesos master log is included. The events logged are the result of a simple reboot of the instance for this particular Mesos slave node.

Should a slave node be temporarily disconnected from the master node, but exceed the timeout period, the Mesos master will forcibly shut down the Mesos slave node. To account for such scenarios, the official Apache Mesos documentation recommends monitoring the Mesos slave process and restarting it if it is terminated for any reason. In this case, this is achieved with a simple check using Monit.

In Listing 2, log events of such a case are listed.

The working Mesos master instance ceases to function

ZooKeeper maintains an active connection to the participants in the quorum, and after a very short timeout lasting less than 10 seconds, it will try to elect a new leading Mesos master node. As long as the number of functional Mesos master nodes is equal to or higher than the quorum size, a new leader will be elected and will replace the unresponsive Mesos master node.

This scenario was tested through a simple reboot of the instance where the leading Mesos master was running. The backup Mesos masters quickly discover the loss of connection to the leading Mesos master and promptly, using ZooKeeper, elect a new leading Mesos master node. The rebooted Mesos master node later joins the cluster as a backup node after coming back online. Quorum size usually corresponds to the smallest number of master nodes that can communicate in order to elect a new leader. To avoid split-brain cluster partitioning, clusters should always have an odd number of master nodes. For example, having three master nodes allows one to be down; having five master nodes allows two to be down, allowing for failure during a rolling update. Additional master nodes can be added for additional risk tolerance. As the size of a cluster grows, it may be desired to increase the quorum size for additional fault tolerance.

```
17:00:26.087030 Disconnecting slave ...5050-5669-S0 at slave(1)@192.168.187.205:5051 (192.168.187.205)
17:00:26.087103 Deactivating slave ...5050-5669-S0 at slave(1)@192.168.187.205:5051 (192.168.187.205)
17:00:26.087155 Slave ...5050-5669-S0 deactivated
17:00:37.940727 Registering slave at slave(1)@192.168.187.205:5051 (192.168.187.205) with id ...5050-5669-S3
17:00:38.116992 Registered slave ...5050-5669-S3 at slave(1)@192.168.187.205:5051
17:00:38.117085 Added slave ...5050-5669-S3 (192.168.187.205)
```

Listing 1 Excerpt from `/var/log/mesos/mesos-master.INFO` showing the deactivation and the new registration of the rebooted slave. Truncated for better readability

```
17:34:23.298998 Shutting down slave ...5050-5669-S3 due to health check timeout
17:34:23.300134 Removing slave ...5050-5669-S3 at slave(1)@192.168.187.205:5051 (192.168.187.205)
17:34:23.301009 Removed slave 20150501-230056-2407081856-5050-5669-S3
17:34:23.536837 Notifying framework ...5050-27030-0006 (marathon) at ...473b-b57a-83121a00a01c@128.39.121.140:43217 of lost slave
↪  ...5050-5669-S3 (192.168.187.205) after recovering
17:34:29.017205 Slave ...5050-5669-S3 at slave(1)@192.168.187.205:5051 (192.168.187.205) attempted to re-register after removal;
↪  shutting it down
17:34:57.329751 Registering slave at slave(1)@192.168.187.205:5051 (192.168.187 .205) with id ...5050-5669-S4
```

Listing 2 Excerpt from `/var/log/mesos/mesos-master.INFO` showing the forced shut down of the Mesos slave process at 192.168.187.205 and the registration as new slave at end. Truncated for increased readability

The setup proposed in this prototype has three Mesos master nodes, with the quorum size set to two. This means that one of the Mesos master nodes can fail without crippling the cluster, as the quorum size dictates the number of election participants that have to be able to communicate to be able to elect a new leader.

An entire region within the hybrid cloud becomes unavailable

If an entire region becomes unavailable, the Mesos nodes located within those regions will, by extension, also become unavailable. In this particular case, the loss of one single site equals the loss of one Mesos master node and four slave nodes. Depending on the type, each node is handled as specified in the test scenarios mentioned above. This was tested by taking down the VPN tunnels at the VPN gateway of the region concerned. This cuts all communication between the affected region and the other ones which is illustrated in Figure 10. The figure illustrates the case where the entire private data center or the public cloud provider might become unavailable due to networking problems. As expected, the Mesos master nodes continued without any issues, as the current leader was not the affected one. As for the affected Mesos slave nodes, after the timeout of 75 seconds, the leading Mesos master node determined that the slave nodes were unresponsive and deactivated them.

The hybrid cloud splits and semi-isolates part of the platform

In the event of a split in the hybrid cloud, resulting in a partly isolated availability region, the quorum mechanics will prevent inconsistencies in the cluster and avoid issues like the split-brain problem.

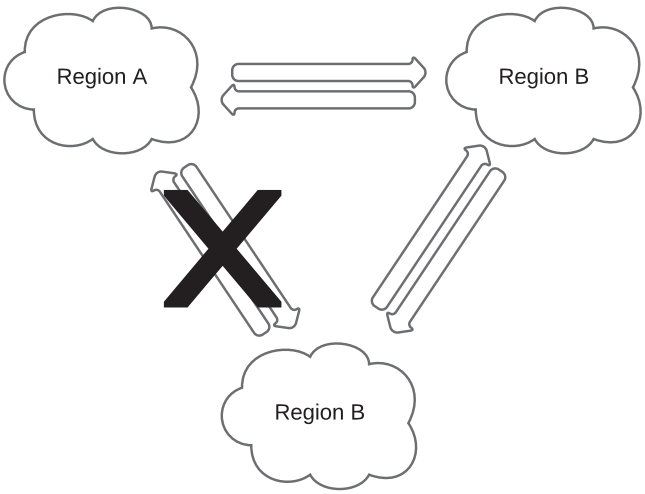

FIGURE 10 Region A cannot reach Region B, but Region B can

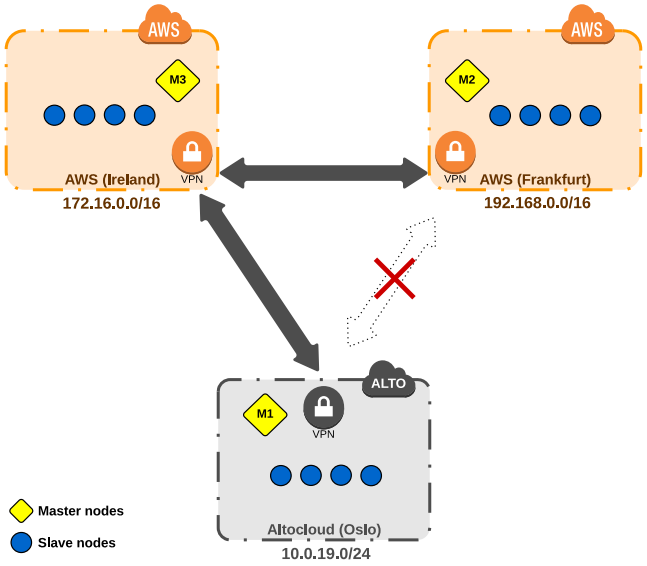

FIGURE 11 An illustration showing what the semi-isolated test scenario looks like

To test this scenario, two simple `iptables` DROP rules were added on the Mesos master node located in Frankfurt with the IP address `192.168.0.5`. The following two lines were executed at the instance:

```
iptables -A INPUT -s 10.0.19.5 -j DROP
iptables -A OUTPUT -d 10.0.19.5 -j DROP
```

The leading master continued with no issues and two other standby Mesos masters correctly redirected to the leading master node. However, after rebooting the ZooKeeper process and Mesos master process on the master nodes, the cluster is unable to elect a new leader. The cluster is effectively frozen as depicted in the screenshot shown in Figure 11. Immediately after the iptables DROP rules were removed, a new leading Mesos master was elected and operations continued as normal.

4.2.2 | Prototype 2: Prioritizing local availability

Due to the architecture of Apache Mesos, it is possible to perform rolling updates on the Mesos master nodes. This ensures no downtime since each Mesos master node is updated separately and then restarted. The setup of this prototype was done by adding two more Mesos master nodes at Altocloud resulting in the final configuration of a five master nodes cluster. Afterwards, the configuration of each of the existing three master nodes was updated and restarted. The cluster was fully operational during this process. As some of the test scenarios do not introduce anything new and behave in the same way as in the previous prototype, a few of those test scenarios have been omitted for this prototype.

An entire region within the hybrid cloud becomes unavailable

The effect on the cluster largely depends on which availability region that becomes unavailable. For any other region than the private site, Altocloud, the effect on the cluster as a whole is limited. For the prototype illustrated in Figure 12, that would mean the loss of a single Mesos

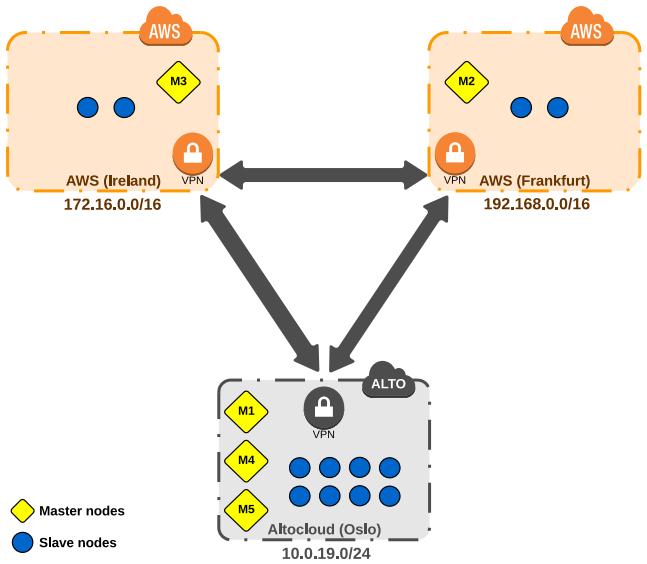

FIGURE 12 Prototype 2: Prioritizing local availability. Focusing on the availability at the local site

master and two slave nodes. With four Mesos master nodes left, and with the majority of the slaves located in Altocloud, the cluster is fully functional with slightly less processing capability. To verify this, the VPN gateway was deactivated at Altocloud, thus simulating both the failure of the private site from the perspective of the Mesos nodes located in the public cloud and the failure of the public cloud Mesos nodes or ISP connection problems from the perspective of the private location.

The Mesos master nodes located in Frankfurt and Ireland entered a leaderless state, waiting for a leader to be elected. Since the Mesos master nodes located at the public cloud locations cannot form a majority, thus satisfying the quorum size requirement, they are in practice frozen while awaiting the connection of one additional Mesos Master node in order to elect a new leader.

While in the private location, Altocloud, the cluster reelects a new leader, as the previous leader was located in Frankfurt and continues to delegate tasks and to restore those that were lost during the disconnection.

The hybrid cloud splits and semi-isolates part of the platform

In contrast to the previous prototype, a semi-split between the regions will not cause the cluster to freeze. This is because the majority of the Mesos master nodes are present at a single location. Ultimately, the Mesos master nodes present at Altocloud will sustain the cluster. However, also for this prototype, should the internal network of Altocloud be split, with semi-isolated Mesos master nodes in combination with a network split between the different regions, it is possible that the cluster will experience the same problem as Prototype 1, with the Mesos master nodes being unable to elect a new leader. It is not immune to the problem although the cluster will still run in the event of a network split occurring between the availability regions.

4.3 | Segmentation of data

Verification of segmentation of tasks and data will be done by confirming that segmented tasks are segmented as specified. For the aspect of privacy, it is difficult to really be sure that traffic is truly private and does not at any time leak out. To verify that no data regarding the tasks are leaked to undesired locations, the source codes of ZooKeeper, Apache Mesos and the used frameworks have to be inspected. To narrow down the scope of this study, a major assumption has been made for the sake of analyzing the aspect of privacy and segmentation.

We shall discuss briefly the flow of the internal traffic of Apache Mesos. Initially, only traffic regarding resource availability at the Mesos slave node is sent to the master in addition to traffic related to the cluster itself and keep-alive pings is assumed to be sent at a regular interval. It is therefore assumed, based on the official documentation of the internal traffic that no information about the available tasks are sent to the Mesos slave nodes before the tasks are being granted.[8] This means that only when tasks are being handed out after checking any constraints will there be any other traffic than the traffic necessary for the cluster to function. Consequently, information about the tasks are sent out on a need-to-know basis. A UML sequence diagram is shown in Figure 13 and describes a possible interaction between the different Apache Mesos components, assuming *MasterY* is the working Mesos master node.

Note that the sequence diagram is simplified and not meant to be fully accurate. It is included to illustrate a possible interaction between the components.

To allow for segmentation of specific data, Mesos slave nodes were tagged with a few attributes after installation. This was done by adding a configuration file in `/etc/mesos-slave/attributes` with the following contents:

```
cloud_type:private;country:norway;city:oslo
```

The contents of this file tag the slaves with attributes marking them with unique properties that can later be used for constraints.

To verify that the segmentation takes place within the hybrid cloud platform, `cloud_type:CLUSTER:private` was added as a constraint for the Marathon application executed. In addition, due to the large amount of resources available, a second constraint was added, `hostname:UNIQUE`, thus requiring a unique hostname for each instance of the task deployed. In short, only one instance of a task can be run on a single slave node, and the slave node has to be tagged with an attribute named `cloud_type`, which is set to `private`.

As there are ten slave nodes located at Altocloud with the attributes set, a maximum of ten task instances can be run at all times, unless additional Mesos slave nodes are added that fulfill the constraints. Figure 14 lists the ten Mesos slave nodes running the constrained Marathon application. Note the subnet of the IP addresses listed, as they are part of the `10.0.19.0/24-subnet` that was allocated to Altocloud.

When attempting to scale further than the ten slave nodes available, the Marathon application is unable to execute additional tasks instances, as there are no available resources that fulfill the constraint requirements. As shown in Figure 15, the Marathon application is unable to scale beyond ten task instances in this particular setup.

4.4 | Automated cloud bursting

The automated cloud bursting solution uses a Python script that was developed as outlined in Section 3. It depends on two configuration `YAML` files and an additional small script for importing some basic functions.

The first configuration file is the main configuration file. It contains the main preferences used by the script. Amazon Web Services API credentials, execution interval, and maximum limits are just a few of those settings. The second configuration file is the launch configuration file, which contains the properties that are used when launching a spot instance.

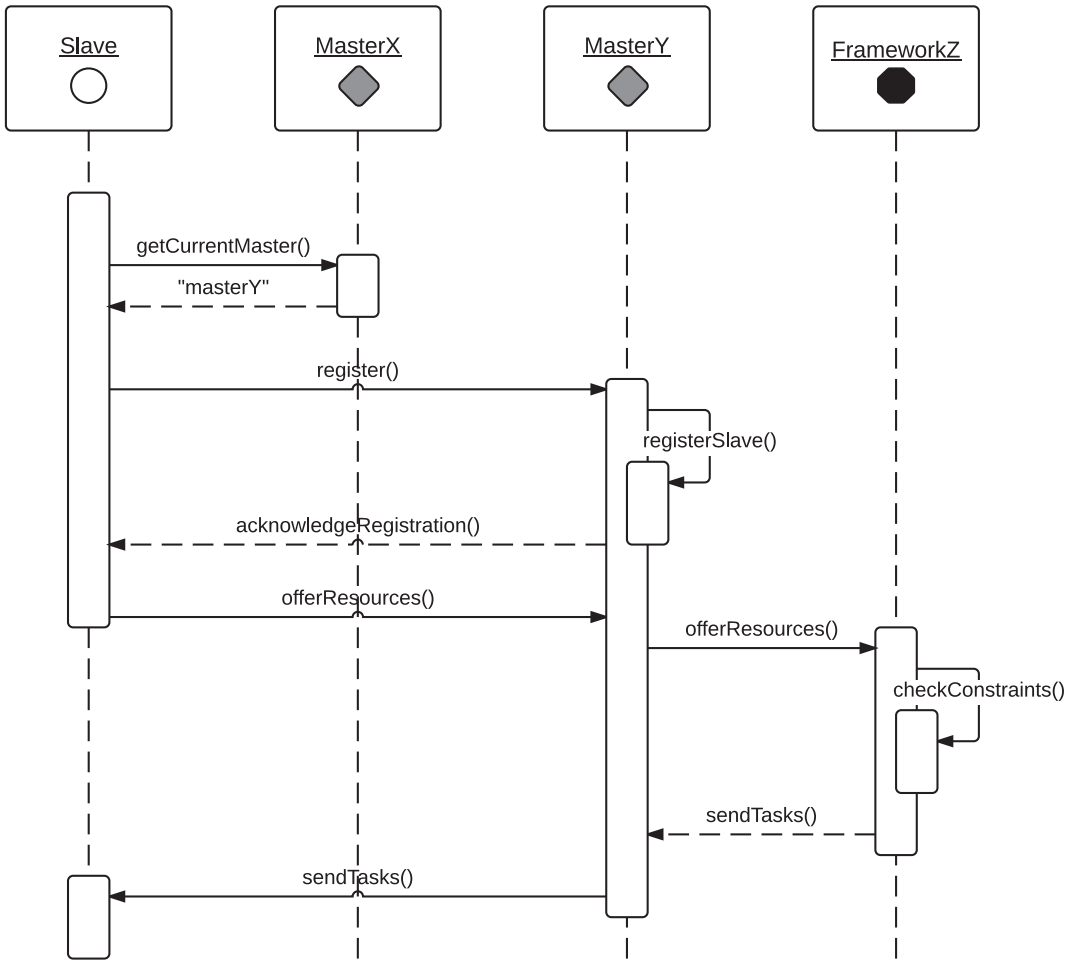

FIGURE 13 A UML sequence diagram describing a possible interaction sequence assuming *MasterY* is the working Mesos master node

ID	Status	Version	Updated
segmented.f9375bdf-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.11:31141	Started	11 minutes ago	6.5.2015, 15.15.32
segmented.f936e6ae-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.8:31963	Started	11 minutes ago	6.5.2015, 15.15.32
segmented.f6ad077c-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.14:31547	Started	11 minutes ago	6.5.2015, 15.15.31
segmented.f6ac6b37-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.12:31691	Started	11 minutes ago	6.5.2015, 15.15.30
segmented.f6ace06b-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.6:31223	Started	11 minutes ago	6.5.2015, 15.15.30
segmented.f6ad2e8d-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.9:31183	Started	11 minutes ago	6.5.2015, 15.15.29
segmented.f6ac9248-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.15:31936	Started	11 minutes ago	6.5.2015, 15.15.29
segmented.f6acb95a-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.7:31471	Started	11 minutes ago	6.5.2015, 15.15.28
segmented.f6ac9249-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.10:31290	Started	12 minutes ago	6.5.2015, 15.15.28
segmented.eee9e086-f3f1-11e4-b3cf-0aaef51ed2d8 10.0.19.13:31981	Started	12 minutes ago	6.5.2015, 15.15.15

FIGURE 14 A screenshot of the Marathon GUI listing the running tasks at the Mesos slave nodes

The particular setup of the hybrid cloud platform followed Prototype 2, as outlined in Figure 12, with the exception of the Mesos slave nodes located at the public cloud providers. The Mesos slave nodes located in Ireland and Frankfurt were deactivated prior to the experiments. The setup utilizes ten Mesos slave nodes as the baseline resources located in the private location, Altocloud. Each of the ten slave nodes was running on a m1.large instance type, resulting in 40 CPUs, 80 GB of memory, and 800 GB of storage in total.

FIGURE 15 A screenshot of the Marathon GUI showing the Marathon application, with the constraint attempting to scale beyond the available resources that fulfill the constraint requirements

The script is designed to be executed as a Marathon application and depends on being present in the cluster, since it attempts to discover the leading Mesos master node dynamically in each iteration. Except for this, the script can be executed like any other Python script from the bash prompt, and it accepts the following arguments:

```
usage: burst.py [arguments]

Arguments:
   --help                        Prints this help message
   -v [--verbose]                Verbose output
   -c [--config]                 Specify another config file
```

The automated cloud bursting solution consists of three main parts: data collection, price, and scaling decision logic, and management of the spot requests and spot instances.

Before any data are collected, the script loads the configuration file into the script and uses the preferences retrieved to resolve a ZooKeeper URL to find the current leading Mesos master. The script then proceeds to collect resources metrics from the leading Mesos master before moving on to collect information about any active spot requests and instances from Amazon Web Services EC2.

If the verbose flag was set when executing the script, the following information about resource usage is displayed in the terminal:

```
|----------------------------------------
| Resource usage: |  Percent     | Count
|----------------------------------------
| CPUs            |  2.68%       | 1.10
| Memory          |  0.25%       | 176 MB
| Disk            |  0.00%       | 0 MB
|----------------------------------------
```

As previously stated, the bidding strategy chosen was the *current price + x*. It was implemented by polling the price in each iteration through the EC2 API. In addition, a maximum bid limit, specified in the configuration file, is checked before returning a bid value. Since the prices fluctuate, so does the bid in the script up to the maximum limit.

The decision on whether to scale or not is determined based on the current resources usage percentage compared to the burst point threshold. The script ensures that the number of spot instances that serves as a Mesos slave node corresponds to the number required to keep the usage percentage just below the burst point threshold. As with the maximum bid limit, there is also a maximum spot slave limit that limits the number of spot slaves the script is allowed to scale up to. With the value set to 0.75, the script will automatically burst into the cloud and start requesting spot instances if the limit of 75% of any single resource is exceeded. The script will evaluate the usage percentage with the current active Mesos slave nodes in addition to the pending resources in order to calculate the percentage. To scale down, the script will calculate the usage percentage of the cluster after removing *x* amount of slaves in order to find the optimal number of spot instances for Mesos slaves with respect to the burst point threshold.

The script will also cancel older spot instance requests that exceed a certain time limit, which is set in the configuration. This is done to take account of spot instance requests that for some reason are stuck and will not result in any spot instances any time soon. This could be due to an error or because the price bid is too low. In the use cases of a cloud bursting solution, the requested resources should appear within a reasonable amount of time. Otherwise, it would not serve its purpose as a cloud bursting solution.

The activity diagram shown in Figure 16 describes the logic that determines the scaling action.

4.5 | Scaling in action

Two experiments have been conducted to showcase the functionality of the cloud bursting solution. Each experiment covered a unique use case that is interesting in a cloud bursting scenario.

To simulate workloads, several Marathon applications were created for the sole purpose of simulating a process hogging available resources. Below is an example of the commands used for resource-hogging tasks.

```
while true; do echo hello world; sleep 60; done
```

As is evident from looking at the command, it does not by itself require a lot of resources to run. The focus of this paper is not on server performance, if this was the case, such a workload could have been simulated using a command like stress. However, each task is allocated a certain

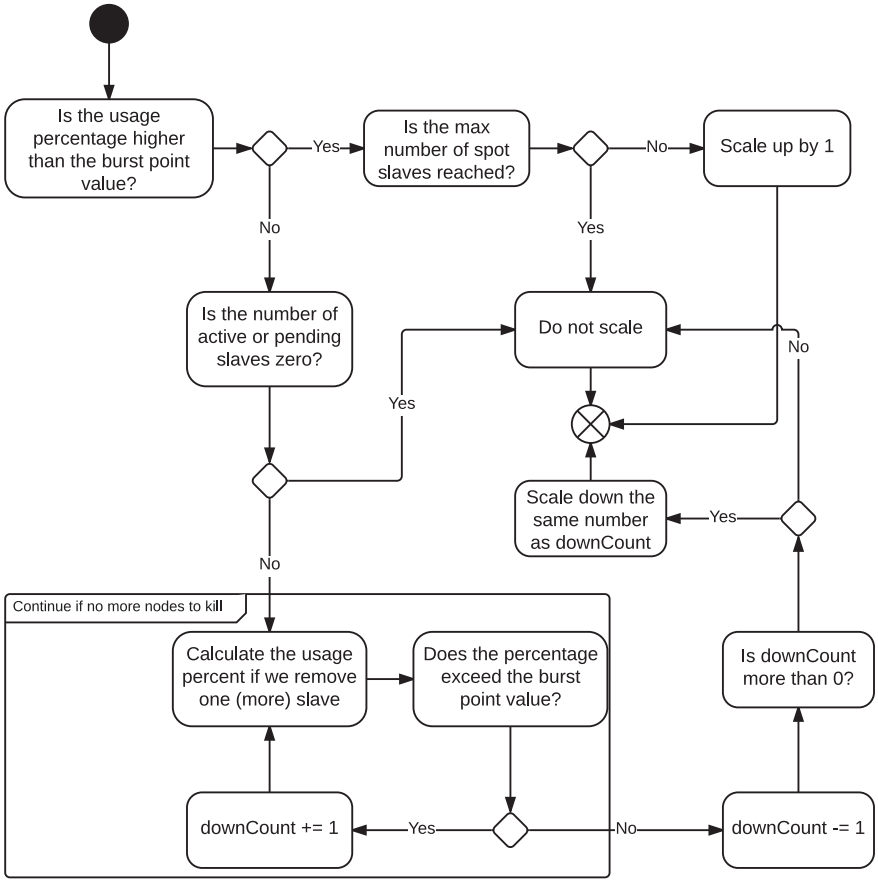

FIGURE 16 An activity diagram showing the decision logic for determining whether or not to scale

amount of resources by the Marathon framework and can within those allocated resources use as little or as much as needed. For the simulations, each of these small `while-loop` tasks is given excessive amounts of resources to quickly hog up large quantities of resources for the simulation.

There are three possible resources variables to simulate:

- CPUs
- Memory
- Disk

CPUs as a resource are normally denoted by integer numbers and they were therefore picked as the resource to exhaust in order to test the cloud bursting solution. To do so, a Marathon application named *hog-cpu-1* was created and scaled up to 100 tasks, effectively hogging 100 CPUs. The baseline resources consist of 40 CPUs and any additional resource required will therefore depend on the resources available through the cloud bursting solution. With the maximum amount of spot slaves set to 10, the cluster cannot accommodate 100 CPUs while scaling up on the lower tier instance types, resulting in a constant need for resources.

4.5.1 | Experiment 1 - Scaling up

Table 6 contains the parameters that were used when conducting Experiment 1. The goal of this experiment is to showcase the cloud bursting functionality, more specifically the scale-up part of the script. At the beginning of the experiment, `hog-cpu-1` will be scaled up to 100 tasks in an attempt to have the script scale up to 10 spot instances to serve as Mesos slave nodes.

As shown in Figure 17, the cloud bursting script scales up the desired number of slave nodes to 10, alternating between 10 and 9. During runtime, the script calculates a bidding price for each iteration, which is the current market price plus a padding of 0.001. However, the number

TABLE 6 The parameters for the cloud bursting experiment 1

Variable	Value
Execution interval	60 seconds
Instance type	m3.medium
Maximum spot slaves	10
Maximum bid limit	0.500
Burst point percentage	85%
Spot request timeout	10 minutes
Price padding	0.001

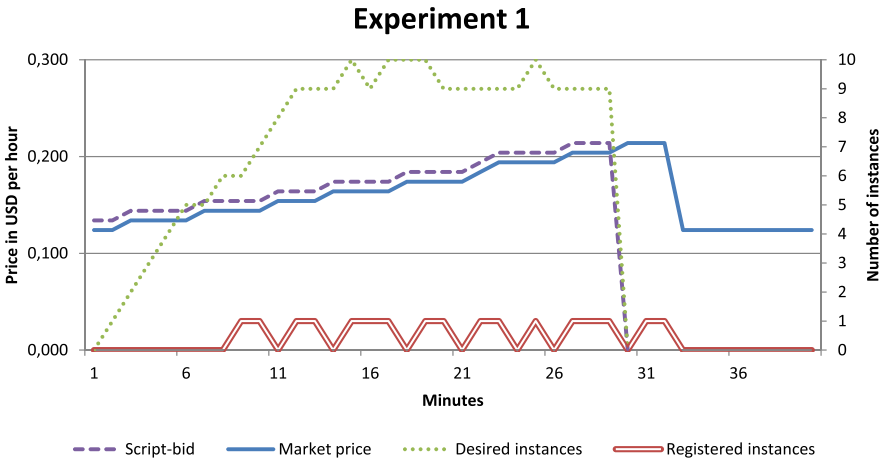

FIGURE 17 Experiment 1: The market price rises at the same interval as the cloud bursting script does and drops as soon as the script stops bidding

TABLE 7 The parameters for the revised cloud bursting Experiment 1 and Experiment 2

Variable	Value
Execution interval	60 seconds
Instance type	c4.large
Maximum spot slaves	10
Maximum bid limit	0.500
Burst point percentage	75%
Spot request timeout	10 minutes
Price padding	0.100

of instances registered to Apache Mesos alternates between one and zero. This is due to the increasing market price, which invalidates the previous spot instance requests, as the bidding price for those instances was lower than the current market price. As a result, the instances are terminated shortly after they are booted up.

After 30 minutes, the CPU-hogging tasks were halted, thus resulting in almost no resource requirements for the cluster. As a result, the number of desired slave nodes, as well as the bidding price, is 0, as no slave nodes are needed. A few minutes later the current market price falls to the original price before the scaling experiment took place.

Experiment 1 revised: Scaling up

As the original Experiment 1 did not showcase the scripts ability to scale up the instances, a number of tweaks were made to the configuration for a revised experiment. As the price changes prevented the script from properly bursting into the cloud within 30 minutes, a revised experiment was deemed necessary. Table 7 lists up the tweaked parameters for the revised experiment*.

Due to the very volatile market price for the `m3.medium` instances type in Frankfurt, the `c4.large` instance type was chosen instead. Compared to `m3.medium`, the market price for `c4.large` instances was far more stable. Additionally, as the goal is to verify the scaling functionality of the script, the price padding was set to 0.100 as an aggressive measure to deal with any spikes in the market price.

As shown in Figure 18, the script managed to successfully request spot instances, with the Mesos slave nodes registering rapidly. After approximately 15 minutes, all the requested slave nodes were online and processing for the cluster. The price remained the same for the entire duration.

4.5.2 | Experiment 2: Scaling down

The goal of this experiment is to showcase the script in a scale-down scenario. The configuration used for Experiment 2 was the same as for the revised Experiment 1, listed in Table 7. This was to ensure that the cloud bursting script would successfully scale down without any major disruptions. Table 8 also contains specific parameters relevant to Experiment 2.

In this experiment, the spot instances that were active in the revised Experiment 1 were used to scale down.

Figure 19 shows the cloud bursting script downscaling the number of spot instances from 10 to 0. The number of `hog-cpu-1` tasks was scaled down from 59 tasks to 24 in intervals of 5, with 24 CPUs being just below the limit when the script will burst into the cloud. As seen in the graph, although the desired number of spot instances is zero, the script keeps them alive for a longer period before scaling down. This is due to the imposed requirement that requires each instance to be alive for at least x number of minutes in an hour cycle, where x is the partial hour limit set in the configuration. For this experiment, the limit was set to 20 minutes to shorten the waiting period for this experiment.

* Please remember that the instances number is integers, although by abuse of representation we do not use bar lines.

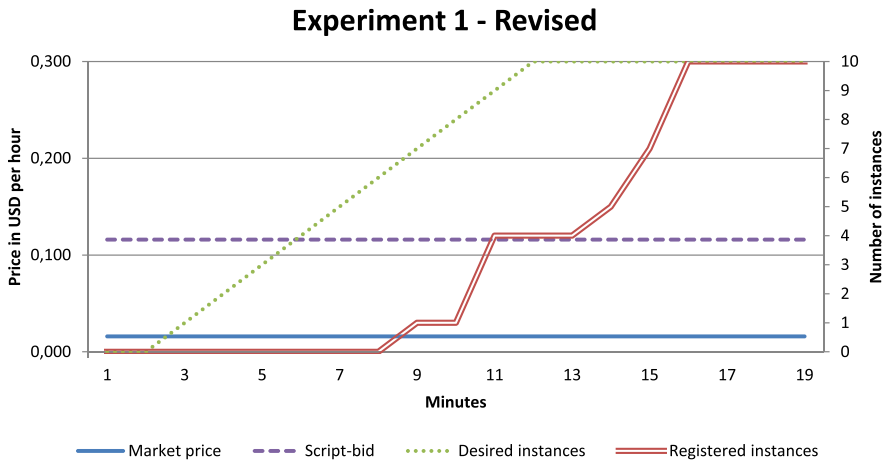

FIGURE 18 Experiment 1 revised: A successful scale-up experiment with the market price being stable for the entire duration of the experiment

TABLE 8 Additional parameters in the configurations set for Experiment 2

Variable	Value
Partial hour threshold	20 minutes

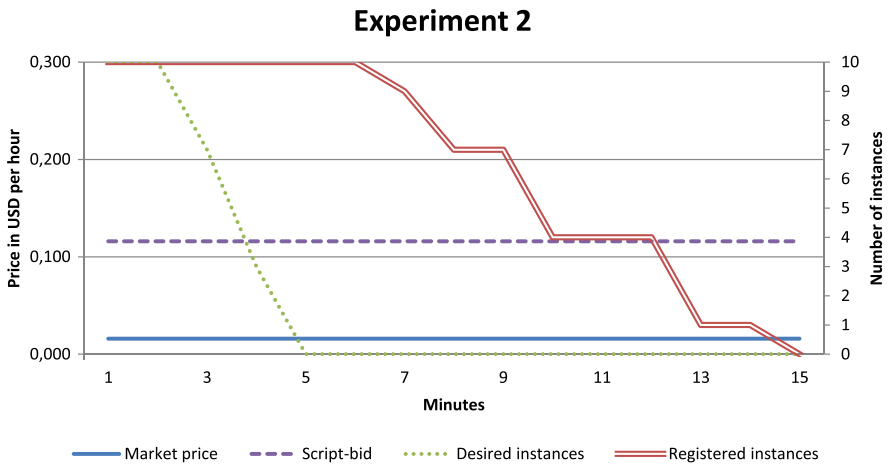

FIGURE 19 Experiment 2: A successful scale-down experiment with the script waiting until the specified minimum time spent in an hour cycle before terminating the instances

Listing 3 is an example of a single iteration of the script. In this case, there are five active spot instances serving the cluster as slave nodes. However, none of them are needed as the current resource usage is below one percent for all three categories of resources. In this case, even the baseline resources available at the private location are mostly idle. The script attempts to terminate the excessive slave nodes. However, since the number of minutes since the last integer hour has not passed the specified limit, the script postpones the termination of the instances. The limit for the fewest number of minutes that is required to pass before terminating an instance is set in the configuration file.

5 | DISCUSSION

This section contains a discussion of the results, the process of the project, additional considerations, and the impact of the findings. The main goal of the paper was primarily to design and implement a hybrid cloud solution. Below the proposed solutions are discussed, each attempting to address the challenges of availability, segmentation, and cloud bursting for a hybrid cloud environment. By abstracting the resources located in heterogeneous cloud environments, practical solutions have been prototyped using Apache Mesos.

5.1 | Hybrid cloud

In this paper, Apache Mesos was used as an abstraction layer between the resources and the higher layers where the applications reside in order to tie together heterogeneous cloud locations. Since Apache Mesos does not support NAT network configurations, VPN was installed. With VPN abstracting the network layer for Apache Mesos, a computer cluster was successfully deployed in a hybrid cloud configuration using private computer resources and public cloud resources.

As mentioned in Section 3, without VPN, the proposed prototypes for setting up the hybrid cloud rely on a vast number of internet routable IP-addresses. When the IPv4 address space is full, it is difficult to acquire the large amount of IPv4 IP addresses that would be required for a larger hybrid cloud platform. These problems can be solved in the future as a result of IPv6 support in OpenStack, Amazon Web Services, and Apache

```
Resolving ZooKeeper url for the working Mesos Master
   Current Mesos master 128.39.121.27
Fetching http://128.39.121.27:5050/metrics/snapshot
   |--------------------------------
   | Current market price:    0.026
   | Maximum bid limit       0.050
   | Our bid                 0.027
   |--------------------------------
   |----------------------------------------
   |       Burst point value set to 0.85
   |----------------------------------------
   |----------------------------------------
   | Resource usage: |  Percent         | Count
   |----------------------------------------
   | CPUs            |  0.22%         | 0.10
   | Memory          |  0.02%         | 16 MB
   | Disk            |  0.00%         | 0 MB
   |----------------------------------------
   |----------------------------
   | Number of:         | Count
   |----------------------------
   | Desired instances  |   0
   | Pending requests   |   0
   | Active instances   |   5
   |----------------------------
Excessive spot instances. Attempting to terminate 5
i-4bf24b8a has not reached the set partial hour limit. 16 minutes have passed.
i-6ff24bae has not reached the set partial hour limit. 9 minutes have passed.
i-84f54c45 has not reached the set partial hour limit. 9 minutes have passed.
i-bff54c7e has not reached the set partial hour limit. 9 minutes have passed.
i-16f24bd7 has not reached the set partial hour limit. 3 minutes have passed.
The script used 0.615634202957 seconds on this loop
Sleeping additional 59.384365797 seconds
```

Listing 3 The verbose output of one iteration in the cloud bursting script. Five instances are excessive and attempted terminated, but this is ultimately postponed due to the lifetime of the instance

Mesos. With IPv6 IP addresses, every node in the cluster can have its own unique Internet routable IP address. IPSEC or another encryption method can be used to secure the communication between the nodes. TLS support for Apache Mesos is upcoming and may be adequate to secure the traffic.

However, depending on the use case, VPN may still be the preferred option, as it isolates and simplifies the network. It may affect the performance; however, it should be investigated whether this is of great importance.

5.2 | High availability

This paper outlines some possible ways of setting up a hybrid cloud using Apache Mesos to achieve higher availability than on a single cloud provider or data center location. By deploying a hybrid cloud in multiple and independent locations, the risks of the entire cluster failing are distributed and reduced as a result of the characteristics of independent probabilities. However, by relying on distributed locations, network becomes a factor in the equation.

This is the reason why two prototypes were designed and evaluated: one prototype that maximizes availability in general, with no concern for where the data is accessed, and another prototype that prioritizes local access.

With Prototype 1, Mesos master nodes are distributed between independent sites, specifically to reduce the failure rates of the Mesos master nodes. By slightly modifying this prototype by adding additional independent sites and additional Mesos master nodes, the failure rates in theory decrease for each node added. However, for each independent site, another dependency is added in the form of dependency on network connectivity, although the failure rates for the network connectivity through the Internet would mainly be regarded as an independent probability separate from the failure rates of the Mesos nodes. This could therefore reduce the probability of failure of the hybrid cloud platform as a whole if the number of sites is sufficient.

Prototype 2 on the other hand prioritizes local availability. Here, the focus is on local access that is independent of external influence. Should Internet access be lost, this setup will allow the cluster to be run and accessed from the network of the private location. This setup considers the

external cloud providers as an additional, and not as a critical part of the cluster. This type of setup works well with cloud bursting use cases, as the private data center functions as a preferred baseline, with the option of utilizing public cloud resources for additional processing power.

Moreover, for the issues discovered regarding semi-isolated networks for Prototype 1 in particular, the availability improvements may not be as great as initially expected. Due to the issue of semi-isolated networks, a three master node cluster may not be sufficient, and additional master nodes may be required in order to partly mitigate this issue. While semi-isolated connectivity problems are not common on the network in general, the use of VPN tunnels makes them a more likely event, since separate VPN tunnels between the sites could in theory fail independently of each other. Consequently, any final setup should allow for a semi-isolated network and the issues that arise. Thus, a three Mesos master node setup may not be good enough for hybrid cloud usage. Five or seven Mesos master nodes should be considered when a semi-isolated network is prone to arise.

5.3 | Data segmentation

Apache Mesos allows for data to be segmented into parts of the hybrid cloud platform using attributes for tagging the Mesos slave nodes and the constraints feature of the Marathon framework. This has also been verified, as no tasks were delegated to any Mesos slave nodes that did not satisfy the constraint requirements. While the solution does segment the tasks as defined, it may not be sufficient for use cases where strict compliance and regulation are the motivation for segmenting data.

However, data segmentation may be interesting when considering task locality, as it may be desirable to place certain services in a particular data center for reasons other than privacy reasons. For example, it may be desirable to group a number of tasks together for performance reasons. Segmenting for this purpose is possible and should be adequate using the Mesos framework Marathon constraints feature.

However, this may not be good enough for every use case that requires segmentation of data, as there may be strict regulations and compliance requirements to satisfy. It could be a requirement that the data do not cross the border, in order to guarantee that the data are not exported outside a nation. In order to properly guarantee data segmentation, source code analysis is required, coupled with auditing tools to verify and attest that the data segmentation is functioning as intended. A self-developed framework that works by using the primitives provided by Apache Mesos is an alternative. This is possible due to the fact that it is built as a two-level scheduler, allowing the individual frameworks to implement the algorithm for deciding whether or not to accept or reject offered resources based on the attributes and resources offered. A framework can therefore be designed to focus on data segmentation with the required functionality and auditing tools to guarantee and attest compliance.

In the end, if security is of the highest order, then the hybrid cloud introduces additional vectors of attack, which is something that needs to be considered. Attack vectors increase with the use of publicly accessible services. VPN does isolate the traffic in transit, although the public cloud providers have direct access to the virtual resources and, by extension, the data contained within them. The terms of service for each provider may contain clauses that give them ownership of the data stored or the right to disclose the information to third parties. These are aspects of public cloud providers that need to be considered before using them with sensitive types of data.

5.4 | Cloud bursting and spot price instances

The cloud bursting solution outlined in this paper automatically deploys additional resources in the public cloud when the resource usage exceeds a prespecified threshold. Additionally, the solution utilizes spot price instances and calculates a bidding price that is based on the current market price and a preset padding to ensure a winning bid.

The solution is scaled to a prespecified burst point threshold that is set in the configuration file. This solution is therefore quite rudimentary and does not account for edge cases, since it will only read the usage percentage and scale on this basis. The reason why such a basic scaling decision algorithm was picked was the lack of appropriate information on which to scale the solution. The Marathon framework for Mesos does provide information about the tasks that are queued and awaiting deployment, although the API does not provide the reason why tasks are queued. This means that the API does not distinguish between tasks that are queued because of resource depletion and tasks that cannot be performed due to constraint requirements.

Moreover, in a multiframework environment, where Hadoop, Chronos, Marathon, or other frameworks are running simultaneously, the script would have to account for the resource usage between all the participating frameworks. This complicates the scaling decision algorithm. This is the reason why the solution scales up in steps and not in one big burst: there is no information that can be used to accurately gauge the required resources, and therefore the required number of spot instance slave nodes. A possible solution to this is to create an algorithm that evaluates the state of the Mesos cluster in combination with the information given by the frameworks as a composition and make a decision. This requires rather complex logic to achieve with the information given by the APIs.

The pricing algorithm implemented is also quite simple and will bid the *current price + x*, where *x* is a prespecified padding set in the configuration. There are several other bidding strategies for building a pricing algorithm, as mentioned in Section 3. For the price bidding strategy to be viable, it has to win a bid within a reasonable amount of time, as the resources are needed when the bids are made. In addition, for the bid to be cost efficient, the proposed algorithm also has to consider on-demand prices.

While the solution does not account for every possible use case, it proves the viability of a cloud bursting solution using spot instances on a hybrid cloud platform using Apache Mesos.

5.5 | Other limitations and considerations

5.5.1 | Performance and abstraction

In this paper, performance has not been evaluated for the hybrid cloud platform. Except from the feasibility of the hybrid cloud setup, performance is perhaps the most important factor to consider when attempting to create such a platform. Due to the way the hybrid cloud platforms may be set up, increased latencies are incurred. The effect of the increased latencies for Apache Mesos is unknown. Apache Mesos benefits from fast networking, and latencies observed on the Internet could possibly have an adverse effect on the performance of the cluster.

The prototypes implemented in this paper use multiple levels of abstraction that each incurs some overhead. Hardware resources were abstracted through the use of virtualization since Altocloud and Amazon Web Services EC2 provide VMs only. If performance is important, one should consider installing Apache Mesos on bare-metal hardware. The network was abstracted and secured with VPN, which also entails some overhead. However, it could argued that the overhead is required, as VPN also secures the traffic. Finally, there is the abstraction done by Apache Mesos. Apache Mesos abstracts the machines, virtual or not, and presents them to applications and frameworks as a pool of available resources.

In addition, the locations of the cloud sites are also important to consider when the performance is vital. Depending on the target audience and the performance requirements, the location of the cloud sites could severely affect the performance of not only the cluster itself but also the perceived quality of service from the perspective of the users of the hybrid cloud.

Before any performance and latency sensitive workloads are considered for such a setup, performance and overheads should be evaluated.

5.5.2 | Depth of testing and experiments

One limitation of this paper is the lack of long-term experiments conducted on availability. Solutions were designed, evaluated, implemented, and analyzed during work on the paper, but no verification beyond theory was carried out. It was assessed that, to get a real indication of actual availability rates for the platform, uptime would have to be monitored over a period of at least a year or more to be close to accurate. This is due to the nature of the metric tied to availability, as it is denoted by a percentage representing availability as a function of time. For instance, at the time of writing, the downtime for Amazon Web Services EC2 for the availability region *eu-central-1* (Frankfurt) and *eu-west-1* (Ireland) was 73 seconds in total during the last year.[45] Due to the high rates of availability, it is difficult to gauge it over a short period of time. For an experiment lasting a few months, it is possible that the availability rate found will be 100%. The opposite is also true. If a cloud provider is unlucky and experiences downtime during an experiment, the resulting availability rate may be terrible for the duration of the experiment but could be impeccable for the rest of the year, resulting in a high availability rate. To properly gauge availability, a long-term experiment has to be conducted, preferably of one to two years duration at least.

Regarding the cloud bursting solution, a few experiments were conducted to verify the main functionality of the script. The main purpose of the experiments was to showcase two use cases that the script is intended to be used for, namely scaling up and down Mesos slave nodes depending on the need for resources. While the experiments were sufficient to verify the feasibility of the proposed solution, the data were only collected for the few experiments outlined in Section 4, and they lack the quantitative foundation required if they are to be used as something more than an indication of the expected behavior of the script. It is not given that the script will handle every use case as outlined in the results, and further experiments must be conducted to verify that the behavior is stable for other use cases.

5.5.3 | DNS management

As no proper configuration and infrastructure was set up for resolving hostnames, the testbed has several limitations. For public facing applications, a proper DNS environment has to be in place for Apache Mesos and Marathon to utilize service discovery for publicly accessible services. With the *Mesos-DNS* project, public services can be made available on the Internet[46] by dynamically binding hostnames to the tasks. However, this requires that the Mesos slave nodes that host those tasks have Internet-routable IP addresses. Otherwise, a load balancer like HAProxy or Nginx has to be set up.

5.5.4 | Spot price instances

There are currently no other providers of the spot price model for instances. The solution developed for automated cloud bursting may not work for other spot price model providers should they emerge, as the principles and mechanisms used may differ considerably. The cloud bursting script is therefore tailored to Amazon Web Services due to a lack of alternative spot price markets. An interesting observation during the cloud bursting experiments was the volatility of the prices for certain instance types. With fewer than ten spot requests, the script was able to push the price upwards from 0.124 at an interval of 0.010 until the experiment was terminated, with the price hitting 0.214 before dropping to 0.124 just a few minutes later. Due to such volatility, alternative instance types, availability regions, instance billing types, or an another cloud provider should be considered if instance stability is important.

5.5.5 | OpenNebula, virtualization and docker

While OpenNebula may has been an adequate solution, it is overly complex for workloads where a simple but powerful data center is desired, because OpenNebula is first and foremost a cloud manager. This entails higher complexity of the code, with multiple services and abstraction being required to facilitate a cloud environment. The rich feature set increases the probability of experiencing issues that affect availability,

as there are many parts that interoperate. By comparison, Apache Mesos is easier to install and manage, and incurs less overhead, since resource isolation with *cgroups* or Docker replaces virtualization. OpenNebula revolves around VMs, providing flexibility and features with this prioritization. On the other hand, Apache Mesos puts applications, tasks, and services at the center and provides flexibility at application level. For a larger-scale data center, this is arguably more important, as it is ultimately the applications and services that provide utility. Moreover, with virtualization, there are arguably two main advantages among many that provide the motivation for using VMs in a data center scenario: *isolation*, commonly referred to as sandboxing, for increased security and to enable multi-tenancy use cases, and, secondly, *resource flexibility and utilization* for increasing resource utilization per VM, as they can be migrated for optimal resource usage on the physical hardware. In addition, resource partitioning offers flexibility, which enables system administrators to shrink, expand, and relocate VMs depending on the requirements.

Docker, an emerging and increasingly popular containerization technology, can be a better alternative for certain workloads than full-fledged VM sandboxing as it has a lighter memory footprint and saves CPU usage. Combined with Apache Mesos, which facilitates application level flexible resource partitioning, the need for virtualization technology decreases for data center scenarios.

5.5.6 | Apache Mesos

Apache Mesos is a relatively new piece of technology with the current version being 0.22.1. Although Apache Mesos is used in production environments in large companies like Twitter and Airbnb, there is a possibility of significant changes to the API and how Apache Mesos functions. Apache Mesos currently only supports Linux environments. For applications requiring other operating systems, Apache Mesos is not an option at present. If there is large enough demand for it, support may arguably be implemented in a future version.

5.5.7 | Suitability

It is important to consider the requirements of the use case for a setup of this size. For hosting a small personal website, installing and managing Apache Mesos may be overkill and might therefore result in far more unjustified complexity. However, there are situations where it may be appropriate to use Apache Mesos, even for low requirement use cases. This could be for educational purposes or to dimension for future scalability, as Apache Mesos is easily scalable after setup. For existing data centers, it may be difficult to migrate to an Apache Mesos cluster due to legacy software with old dependencies or non-Linux applications. It is important to consider the cost of migration in both time and resources compared to the potential benefits. Migrating existing solutions may be a large undertaking depending on their size and complexity.

5.6 | Future work and improvement suggestions

This section contains suggestions for improvements and further work that will be interesting to investigate.

Evaluating the performance and long-time availability of a hybrid cloud setup

In this paper, a hybrid cloud solution with automated cloud bursting has been designed and implemented. With a working setup, what remains to be done is to gauge the long-time availability and performance of the proposed prototypes. The results of such investigations in combination with this paper would make it possible to determine the level of utility the proposed solutions has. Another aspect that would be interesting to investigate is quality of service. This has to be gauged by monitoring the services running on the hybrid cloud platform and comparing them to the expected quality of service, for instance the expected response time, latency, performance, or any other metric that affects the quality of the service.

Evaluate and prototype additional public cloud providers for a hybrid cloud platform

While the different availability zones for Amazon Web Services functioned as different cloud locations in the paper, additional public cloud providers other than Amazon Web Services were not considered. Other cloud providers may use different architectures, locations, hypervisors, and hardware that in theory should increase the availability if used in the hybrid cloud platform, as the degree of independence increases. However, for the proposed solution, this would require the other public cloud providers to facilitate the setup of a site-to-site VPN solution.

The possibilities of using additional public cloud providers to build a hybrid cloud platform would be interesting to investigate and evaluate further.

Improve the cloud bursting solution

There are many interesting improvements that could be made to the automated cloud bursting script. As regards the price bidding algorithm for spot instances, a new algorithm should be developed that is far more resistant to price spikes and that also considers on-demand prices. For instance, when the spot prices exceeds the on-demand prices, the script could choose to boot up an on-demand instance instead of a spot price instance.

Another improvement to the script is to develop a new or expand the existing scale decision logic to rely on more tangible data when a scaling decision is made. By finding more suitable data to aid the decision to scale, the script could more intelligently request the required number of nodes directly, instead of scaling one node per script iteration.

Sometimes, a lower tier instance type is more expensive than a higher tier instance type solely because of demand. This was observed during work on this paper. An algorithm could therefore be developed that picks the most price efficient instance type based on the spot market prices for each instance type normalized with respect to the amount of resources on the VM. With such an algorithm, the most price efficient instance type for the required amount of resources can be requested without being limited to one instance type. Another approach[47] focuses on such a concept and uses a theater analogy to evaluate and pick a suitable VM given certain performance and pricing requirements.

Although there are no spot price market competitors among the other public cloud providers other than Amazon Web Services, the pricing for the on-demand instances differs. While these prices are fairly static, the cloud bursting solution could evaluate the pricing for other cloud providers as well, and pick the most suitable option depending on current availability and resource needs. Furthermore, as a future work, we aim to test containers. In fact, it is advantageous to use containers for a cloud bursting solution as they have a faster boot-up time than normal VMs which take few minutes typically to be provisioned. Therefore containers can provide faster resource adaptiveness to variations in the traffic. Moreover containers enjoy smaller memory footprint and CPU usage compared to VMs.

Using ratio price over performance to optimize cost

Most cloud bursting solutions only consider instance prices when performing the scaling. However, the ratio price over performance is a more appropriate concept as more expensive instances do not necessarily provide higher performance. Since VMs come with different flavors in a public cloud, a sophisticated decision making mechanism for performing the scaling need to be developed so that to provision the right number and combination of instances that meet the SLA requirements. For private clouds, it is possible to rather consider the ratio energy over performance as in contrast to public clouds, the price is not an observable cost in private cloud.[†]

6 | CONCLUSION

This paper presents multiple prototypes for setting up a hybrid cloud platform using Apache Mesos to weave together heterogeneous cloud types and geographical locations into a unified platform. Each of the proposed prototypes focuses on a specific perspective, maximizing availability and local access prioritization.

Data segmentation has also been demonstrated in this paper, with the hybrid cloud platform using Apache Mesos and the framework Marathon to set constraints to segment the data flow. However, for strict compliance requirements or a high level of confidentiality requirements, the outlined solution may not be adequate. Alternative solutions have been proposed.

An automated cloud bursting solution has also been prototyped and implemented on the hybrid cloud platform. It automatically requests spot instances in Amazon Web Services EC2 depending on resource usage on the hybrid cloud platform. The solution calculates a bidding price derived from the current market price and that adapts to price fluctuations. When resource usage decreases, the solution considers billing mechanics to ensure that resources that have been paid for are fully utilized for maximum economic efficiency before terminating excessive resources.

While the paper presents functional and viable solutions with respect to availability, segmentation, and automated cloud bursting for a hybrid cloud platform, further work remains to be done to improve and confirm the proposed solution, in particular a performance analysis of the proposed solutions.

ORCID

Anis Yazidi https://orcid.org/0000-0001-7591-1659

REFERENCES

1. Google Inc. Google Trends - web search interest: multi cloud, hybrid cloud, inter cloud - worldwide. 2019. https://trends.google.com
2. Xue N, Haugerud H, Yazidi A. Towards a hybrid cloud platform using Apache Mesos. In: *Security of Networks and Services in an All-Connected World: 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, July 10-13, 2017, Proceedings*. Cham, Switzerland: Springer; 2017:143-148.
3. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J. Large-scale cluster management at Google with Borg. In: Proceedings of the 10th European Conference on Computer Systems; 2015; Bordeaux, France.
4. Schwarzkopf M, Konwinski A, Abd-El-Malek M, Wilkes J. Omega: flexible, scalable schedulers for large compute clusters. In: Proceedings of the SIGOPS European Conference on Computer Systems (EuroSys); 2013; Prague, Czech Republic.
5. The Apache Software Foundation. Apache Mesos. 2019. Accessed August 15, 2019. http://mesos.apache.org
6. Leopold G. Apache Mesos emerges as datacenter OS. 2019. Accessed August 15, 2019. http://www.enterprisetech.com/2015/02/23/apache-mesos-emerges-as-datacenter-os
7. Zaharia M, Hindman B, Konwinski A, et al. The datacenter needs an operating system. In: Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing; 2011; Portland, OR.

[†]We are grateful for an anonymous reviewer who suggested considering the ratio price over performance as a future research direction.

8. Hindman B, Konwinski A, Zaharia M, et al. Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation; 2011; Boston, MA.
9. The Apache Software Foundation. 2019. Accessed August 15, 2019. http://mesos.apache.org/documentation/latest/powered-by-mesos/. Powered By Mesos
10. Twitter, Inc. Mesos graduates from Apache incubation. 2013. Accessed August 15, 2019. https://blog.twitter.com/2013/mesos-graduates-from-apache-incubation
11. Vavilapalli VK, Murthy AC, Douglas C, et al. Apache hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th ACM Annual Symposium on Cloud Computing (SoCC '13); 2013; San Jose, CA. http://doi.acm.org/10.1145/2523616.2523633
12. The Apache Software Foundation. Apache myriad. Accessed August 15, 2019. http://myriad.incubator.apache.org
13. Liu F, Tong J, Mao J, et al. NIST cloud computing reference architecture. *NIST Spec Publ.* 2011;500(2011):1-28.
14. Nair SK, Porwal S, Dimitrakos T, et al. Towards secure cloud bursting, brokerage and aggregation. In: Proceedings of the 2010 8th IEEE European Conference on Web Services (ECOWS); 2010; Ayia Napa, Cyprus.
15. Connor TR, Southgate J. Automated cloud brokerage based upon continuous real-time benchmarking. In: Proceedings of the 8th International Conference on Utility and Cloud Computing; 2015; Limassol, Cyprus.
16. Breiter G, Naik VK. A framework for controlling and managing hybrid cloud service integration. In: Proceedings of the 2013 IEEE International Conference on Cloud Engineering (IC2E); 2013; Redwood City, CA.
17. Ardagna D, Di Nitto E, Casale G, et al. Modaclouds: a model-driven approach for the design and execution of applications on multiple clouds. In: Proceedings of the 4th International Workshop on Modeling in Software Engineering; 2012; Zurich, Switzerland.
18. Rahabok I. *VMware vRealize Operations Performance and Capacity Management*. Birmingham, UK: Packt Publishing; 2014.
19. VMWare, Inc. Cloud computing. 2019. Accessed August 15, 2019. http://www.vmware.com/cloud-computing/hybrid-cloud.html
20. Butler B. Re-examining cisco intercloud strategy. Network World. 2019. Accessed August 15, 2019. http://www.networkworld.com/article/2864857/cloud-computing/re-examining-cisco-s-intercloud-strategy.html
21. Coyne L, Dain J, Forestier E, et al. *IBM Private, Public, and Hybrid Cloud Storage Solutions*. IBM Redbooks. 2018.
22. Rackspace, Inc. Hybrid cloud computing, hybrid hosting by rackspace. 2019. Accessed August 15, 2019. http://www.rackspace.com/cloud/hybrid
23. Bubak M, Baliś B, Kitowski J, et al. Paasage: model-based cloud platform upperware. 2019. Accessed August 15, 2019. http://www.paasage.eu
24. Moreno-Vozmediano R, Montero RS, Llorente IM. Multicloud deployment of computing clusters for loosely coupled MTC applications. *IEEE Trans Parallel Distrib Syst*. 2011;22(6):924-930.
25. Ghodsi A, Hindman B, Konwinski A, Zaharia M. Mesosproposal. 2010. Accessed August 15, 2019. http://wiki.apache.org/incubator/MesosProposal
26. Bittencourt LF, Madeira ERM, Da Fonseca NLS. Scheduling in hybrid clouds. *IEEE Commun Mag*. 2012;50(9).
27. Borja S, Ruben MS, Ignacio MT, Ian F. An open source solution for virtual infrastructure management in private and hybrid clouds. *IEEE Internet Comput*. 2009;1:14-22.
28. Kovács Á, Lencse G. Modelling of virtualized servers. In: Proceedings of the 2015 38th International Conference on Telecommunications and Signal Processing (TSP); 2015; Prague, Czech Republic.
29. Guo T, Sharma U, Shenoy P, Wood T, Sahu S. Cost-aware cloud bursting for enterprise applications. *ACM Trans Internet Technol.* 2014;13(3). Article No. 10.
30. Feller E, Rilling L, Morin C. Snooze: a scalable and autonomic virtual machine management framework for private clouds. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012); 2012; Ottawa, Canada.
31. Razavi K, Ion A, Tato G, et al. Kangaroo: a tenant-centric software-defined cloud infrastructure. In: Proceedings of the 2015 IEEE International Conference on Cloud Engineering; 2015; Tempe, AZ.
32. Williams D, Jamjoom H, Weatherspoon H. The Xen-Blanket: virtualize once, run everywhere. In: Proceedings of the 7th ACM european conference on Computer Systems; 2012; Bern, Switzerland.
33. Bicer T, Chiu D, Agrawal G. A framework for data-intensive computing with cloud bursting. In: Proceedings of the 2011 IEEE International Conference on Cluster Computing; 2011; Austin, TX.
34. Bicer T, Chiu D, Agrawal G. Time and cost sensitive data-intensive computing on hybrid clouds. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012); 2012; Ottawa, Canada.
35. Aktas MS. Hybrid cloud computing monitoring software architecture. *Concurrency Computat Pract Exper*. 2018;30(21);e4694.
36. Huedo E, Moreno-Vozmediano R, Montero RS, Llorente IM. Architectures for enhancing grid infrastructures with cloud computing. In: *Grids, Clouds and Virtualization*. London, UK: Springer; 2011:55-69.
37. Chu H-Y, Simmhan Y. Cost-efficient and resilient job life-cycle management on hybrid clouds. In: Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium; 2014; Phoenix, AZ.
38. Fadel AS, Fayoumi AG. Cloud resource provisioning and bursting approaches. In: Proceedings of the 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing; 2013; Honolulu, HI.
39. De Assunção MD, Di Costanzo A, Buyya R. 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; 2009; Garching, Germany.
40. Peláez V, Campos A, García DF, Entrialgo J. Online scheduling of deadline-constrained bag-of-task workloads on hybrid clouds. *Concurrency Computat Pract Exper*. 2018;30(19):e4639.
41. Voorsluys W, Buyya R. Reliable provisioning of spot instances for compute-intensive applications. In: Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA); 2012; Fukuoka, Japan.
42. Xue N, Haugerud H, Yazidi A. Automated cloud bursting. Accessed August 15, 2019. https://github.com/haugerud/cloud-burst
43. Apache Software Foundation. Mesos crashes if any configured zookeeper does not resolve. Accessed August 15, 2019. https://issues.apache.org/jira/browse/MESOS-2186
44. Apache Software Foundation. C client bug in zookeeper_init (if bad hostname is given). Accessed August 15, 2019. https://issues.apache.org/jira/browse/ZOOKEEPER-1029

45. CloudHarmony. CloudSquare service status. Accessed August 15, 2019. https://cloudharmony.com/status-1year-of-compute-group-provider
46. Kakadia D. *Apache Mesos Essentials*. Birmingham, UK: Packt Publishing; 2015.
47. Borgenholt G. *Audition: A DevOps-Oriented Quality Control and Testing Framework for Cloud Environments* [master's thesis]. Oslo, Norway: Univeristy of Oslo; 2013.

How to cite this article: Haugerud H, Xue N, Yazidi A. On automated cloud bursting and hybrid cloud setups using Apache Mesos. *Concurrency Computat Pract Exper*. 2020;32:e5662. https://doi.org/10.1002/cpe.5662