Effects of reducing VMs management times on elastic applications

Jose A. Pascual[1], Jose A. Lozano[2], and Jose Miguel-Alonso[3]

[1]University of the Basque Country UPV/EHU
[3]University of the Basque Country UPV/EHU
[2]Basque Centre for Applied Mathematics (BCAM), University of the Basque Country UPV/EHU

June 5, 2018

Abstract

Cloud infrastructures provide computing resources to applications in the form of Virtual Machines (VMs). Many applications deployed in cloud resources have an elastic behavior, that is, they change the number of servers (VMs) dynamically, adapting the application to the workload. Scaling-out and scaling-in operations are managed by an auto-scaler module, which can be reactive (adapting the number of VMs to the current workload) or proactive (adapting to the expected future workload). The cloud infrastructure provides a management interface to create (deploy) and destroy (shutdown) server instances, operations that require some time to complete. In this work we evaluate to what extent the reduction of the time required by VM management operations, namely deployment and shutdown, impacts the performance of applications and the behavior of reactive and proactive auto-scaling policies. After establishing several ideal boundaries on the use of resources, we carry out a set of experiments that show how short management times drastically reduce the use of resources, while allowing the application to operate within the required performance bounds.

1 Introduction

Cloud computing has emerged as a new paradigm to host and execute different types of applications. A main difference between cloud computing and other classical computing approaches is the virtualization of resources, which provides great benefits to both the owner and the user of the infrastructure. In particular, the resources hosted by physical servers are provided in the form of abstract computing units that are implemented as Virtual Machines (VMs).

One of the characteristics that makes cloud infrastructures so appealing for users is that they allow the *on-demand* utilization of resources, that is, VMs for a running application can be acquired and released dynamically, enabling the implementation of elastic applications [24] [29]. Furthermore, users pay according to a scheme known as *pay-as-you-go*, meaning that they are only billed for the resources they actually reserve. The relationship between the tenants (users deploying applications) and the cloud provider (the one managing the infrastructure) is sealed by a Service Level Agreement (SLA) that is defined in terms of one or several Service Level Objectives (SLO).

Virtualized data centers support the execution of a large variety of applications, from scientific codes to web applications. Each of these applications is initially deployed with a fixed number of servers (implemented as VMs) in order to serve end-user requests. End-users normally expect that their requests will be processed within the limits established by another SLA signed with the application provider. It is important to differentiate this SLA, which we will call Application-SLA (A-SLA), from the one signed between the application owner and the cloud infrastructure, which we will call, from now on, Infrastructure-SLA (I-SLA).

As the input workload generated by end-users can be very variable, the number of application instances (VMs) used by the application should adapt correspondingly, to avoid A-SLA violations caused by saturated resources. Similarly, if the number of VMs is excessive, some resources should be released to avoid unnecessary costs. This resource-to-workload adaptation task could be performed manually or, preferably, carried out by an auto-scaler module that, without human intervention, adjusts the resources assigned to the application. The auto-scaler checks periodically a set of metrics that measure input workload and resource utilization. It may then determine that some VMs should be added to avoid A-SLA violations, or that it is safe to release some VMs if they are no longer necessary, thus triggering a scaling event. Note that we are dealing with *elastic* applications and *horizontal* auto-scaling [31].

Scaling events are implemented using the management interface offered by the cloud provider to acquire and release resources. Deploying a new application instance is not instantaneous: a newly requested VM needs some time (several minutes [21]) until it is fully operational and can start processing end-user requests. Therefore, many A-SLA violations may happen while waiting for the new VM. A proactive auto-scaler takes this time into consideration, trying to anticipate future user demands. Removing a VM is not instantaneous either: the VM must process all the pending requests, and then shut down the services. Only then it is possible to remove it safely. Notice that auto-scaling decisions do not result in immediate changes in service times (time required to process the requests); for example, some time is required to have a newly deployed VM effectively processing requests and, thus, reducing the workload of other VMs.

In this paper we explore the benefits of having very short, even instantaneous, deployment and release operations. This is a complex task to be carried out by the cloud provider, and we do not explore here how to do it – this is left as future work. However we demonstrate how a shortening of these resource

management operations affects the behavior of elastic applications considerably, allowing them to use fewer resources without putting A-SLA compliance at risk. We also demonstrate how, when management times are short, the performance difference between simple reactive auto-scalers and their proactive counterparts is remarkably reduced, avoiding the need to use and tune sophisticated prediction algorithms.

This study has been carried out using a large number of simulations. We model an elastic application hosted in a cloud infrastructure, under the control of different auto-scalers. We use actual web traces to emulate the input received by the application. As this work is not focused on evaluating auto-scalers, we had to narrow the number of target metrics and the number of auto-scaling algorithms evaluated. We have chosen a representative instance of reactive auto-scalers and another one of proactive auto-scalers, both working with a single target metric, as most actual auto-scalers do. Results show that both make better decisions when managed times are reduced.

The rest of the paper is organized as follows. In Section 2 we define the context and motivation of this work, following in Section 3 with a description of the type of elastic applications considered in this study. We continue in Section 4 by presenting the simulation-based experimental framework, including a description of the set of experiments carried out. Section 5 is devoted to establishing a baseline to compare the results presented in sections 6 and 7, in which the results of the experiments are discussed. Next, in Section 8 we discuss some previous works about shortening resource management times of cloud applications. The paper finishes in Section 9 with some conclusions and an enumeration of future lines of work.

2 Context and motivation

In this section we put in context the work presented in this paper. We start by describing the cloud computing scenario in which elastic applications run. Then we analyze the lifetime of a VM, in order to understand the importance of shrinking management times. We end this section with a brief summary of the main contributions of this work.

2.1 Cloud computing environments

A cloud computing environment can be divided into three layers: the infrastructure, the tenants and the end-users, as depicted in Figure 1. The right side of the figure represents the manager of the infrastructure, which offers resources to their clients, the tenants that will pay for using them. Applications managed by the tenants (represented in the middle of the figure) acquire or release resources as needed in order to provide a set of services to groups of end-users (left-hand side of the figure). An I-SLA governs the relationship between provider and tenant. A different A-SLA governs the relationship between the tenants and the end-users.

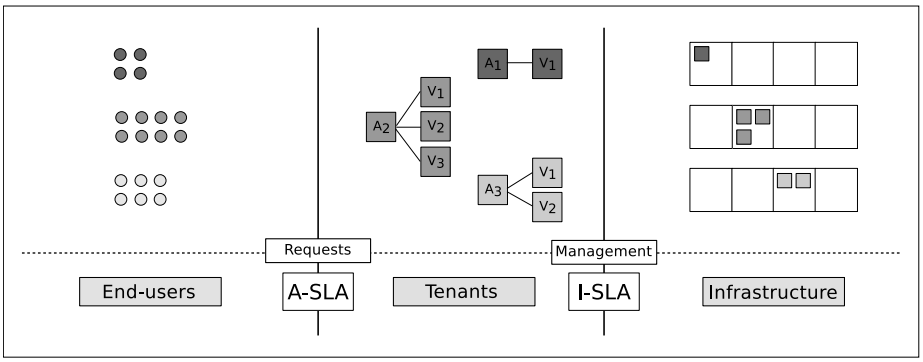


Figure 1: Parties involved in cloud-deployed applications. The infrastructure provides virtualized resources such as VMs (right) to tenants that run applications (middle). Applications deployed in the cloud provide services to end-users (left). These end-users generate requests that must be served by the application. Both tenants and infrastructure must meet the restrictions established in the SLAs signed with their respective customers (A-SLA and I-SLA).

A fundamental aspect of the I-SLA is the price of the resources, but it may include other factors, such as resource availability and also the management times of VMs: the infrastructure must implement management operations within some time bounds. The A-SLA normally includes an acceptable maximum response time, that is, the maximum time since a user request is received by the application until the reply is sent. When this time is exceeded, we say that an A-SLA violation has occurred.

We focus on horizontal scaling [31], that is, on applications that can be scaled out (adding new server instances) or scaled in (removing server instances) in order to adapt resources to demand. The application consists, thus, of a collection of identical servers that receive requests to perform tasks from the end-users. These requests arrive at the front-end or load balancer that distributes them among the active servers. In this context, we will interchangeably use the words VM and application server, as one is the way of implementing the other in a cloud infrastructure. As, in this work, we do not study the role of the load balancer (we focus on the collection of servers), we assume that a proper mechanism is in place to inform the load balancer about the availability of a new VM, and about the need to stop redirecting requests to a VM that is to be removed.

Additionally, we consider that elasticity is implemented by means of an auto-scaler module implementing a certain scaling policy. Its role is to react to changes in the input request rate, making decisions about adding or releasing resources. These decisions are implemented, through some invocation to the management interface, by the infrastructure, which deploys or shuts down VMs within the time limits established by the I-SLA. The auto-scaler tries to minimize the use of resources (to save costs to tenants) while guaranteeing A-SLA

compliance (to provide a satisfactory service to end-users).

Recently the use of containers as a new model for resource management in cloud environments has become increasingly popular [12]. Throughout the rest of the paper we will focus on VMs although all the analysis and conclusions about auto-scaling remain the same for container-based elastic applications, as deployment and release of containers also require non-null management times.

2.2 VMs lifetime

As explained in the previous subsection, when the auto-scaler decides that new VMs are required, these are requested to the infrastructure through a management interface. However, those VMs are not deployed instantly. In Figure 2 we have represented the lifetime of a VM, and we can see how a Deployment Time (T_D) is required before a VM is ready to serve requests. The operations to be carried out during this time include locating the adequate resources, booting the operating system, setting up the required services (network, storage, etc.) and finally starting up the application itself. Actual values of T_D may vary broadly, depending on many factors that include, among others, the particular OS being used, and the location of the VM image to use: stored locally or somewhere else in a network-based storage.

Only after the VM has been deployed, can it start serving requests. The time that VMs spends ready to accept end-user requests is called Active Time (T_A). A VM will be in this state until the auto-scaler decides that it is no longer necessary. When this happens, the VM must be released, a procedure to be completed in several steps and that also requires some non-negligible Shutdown Time (T_S). The time required to safely release a VM is not easy to bound. First, it is necessary to complete all the queued requests and those being processed by the VM – if any. The duration of this stage is called the Backlog Time (T_B) and depends on the number of pending requests. The procedure to totally shut down the VM, releasing the corresponding resources, can only start when the VM is totally idle, and takes a certain Release Time (T_R). When a VM is in the backlog processing phase, the auto-scaler may reclaim it. Then the shutdown process is canceled and the VM will never enter the release phase, returning directly to the active state. Recycling a VM in the backlog stage is, thus, more efficient than releasing a VM and acquiring a new one.

Clearly, two of the times depicted in Figure 2 depend on characteristics (and the I-SLA) provided by the cloud infrastructure: T_D and T_R are normally fixed, due to technological restrictions. The value of these management times can vary greatly between different cloud providers, but they are usually in the range of minutes for T_D, and tens of seconds for T_R [21].

As a side-effect of non-instantaneous management operation is the need of auto-scalers to implement a cool-down period, longer than the management times: after making a scaling decision, new ones are suspended, waiting for the effects of the last one committed. This imposes a limit to the adaptability of the application.

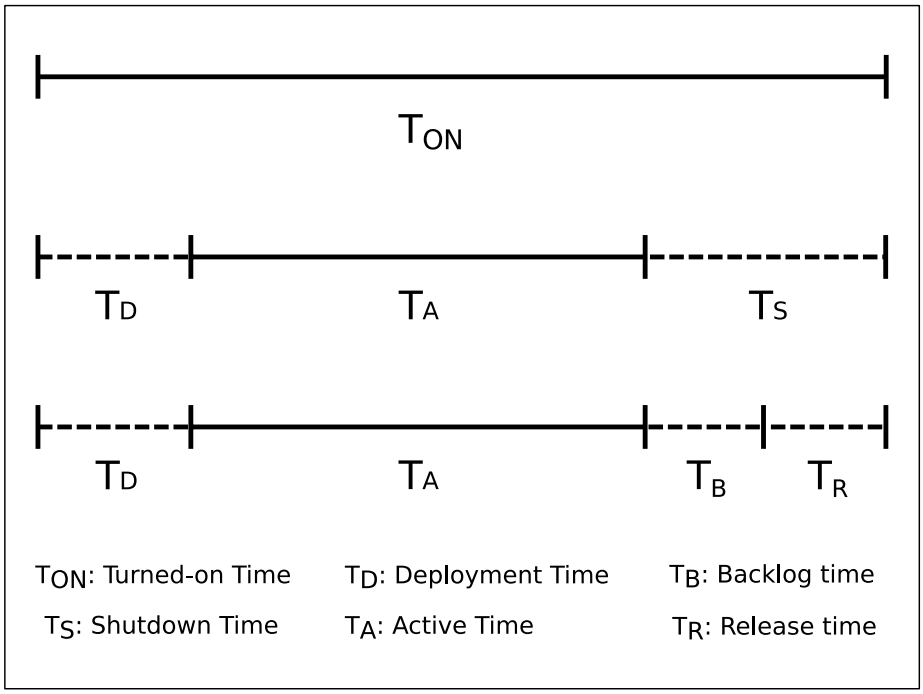


Figure 2: Dissection of the lifetime of a VM.

2.3 Expected effects of reducing VM management times

From the analysis carried out in the previous section, it should be clear that the time that application instances spend serving requests depends mainly on the decisions made by the auto-scaler. However, there are some additional, management-related overheads, whose duration depend on the specifics of the cloud provider, and change from one to another.

Once the auto-scaler of an application has decided to add or remove VMs from the allocated pool, the infrastructure must deploy them in, at most, T_D or release them in, at most, T_R, as determined by the I-SLA. We expect these times to have a negative effect on cost and/or performance but, can we measure this effect? What are the benefits of shrinking these times as much as possible? Clearly, management times close to zero allow the applications to be more reactive, as the decisions of the auto-scaler have an impact on request service time almost immediately. As a result of this, these decisions would be more effective, and this should translate into a reduction in the amount of resources required by the applications to meet the A-SLA. Subsequently, proactive auto-scaling policies that try to counteract the effects of those delays would be less necessary, making the use of simpler reactive policies worthwhile.

Moreover, if the infrastructure and, thus, the application could react faster to resource changes, the frequency of metric checks and auto-scaling decisions could

be increased accordingly, as actions result in noticeable performance changes more rapidly.

The three main contributions of this work are as follows:

1. An in-depth analysis on the effects of VM management times on the performance of elastic applications.
2. A comparison of reactive and proactive auto-scaling policies under scenarios of long and short VM management times.
3. An analysis of the impact of the frequency of auto-scaling decisions on the performance of elastic applications, again taking into consideration long vs. short VM management times.

Notice that in this paper we evaluate what would happen if cloud providers had adequate mechanisms to enable the reduction of VM management times, but an analysis of these mechanisms falls outside the scope of this paper and is left as future work. In Section 8 we discuss some proposals found in the literature.

3 Modeling elastic applications

In this section we describe the architecture of the class of elastic applications used in this work, as well as the specifics of the auto-scaler that is in charge of triggering scaling events.

3.1 Architecture of the application

We consider applications using a two- or three-tier architecture:

1. A load balancer (LB) receives the requests generated by end-users and redirects them to any of the identical servers (VMs) that compose the application. The LB may be implemented in a hardware device or as a DNS-based redirection. We do not model it in this work; it is simply assumed that the LB distributes requests evenly, and that it provides an interface to enroll and withdraw application servers.
2. A business tier that implements the logic of the application. It is composed of a variable collection of identical VMs that process input requests (from end-users, redirected by the LB) and generate the required responses.
3. An optional persistence (storage) tier that implements the database requirements of the application.

Although the study performed in this work is applicable to any kind of applications supporting horizontal scaling [31] [14], for the sake of simplicity and due to the fact that we do not model intra-application communications,

we have decided to use only 2-tier applications. The number of servers in the business tier is managed by an auto-scaler; it will make scaling decisions using a given scaling policy. This component and the policies used in this work are described next.

3.2 Auto-scaling policies

The auto-scaler is the component in charge of deciding whether to increase or decrease (scale) the number of servers of an application. These decisions are performed at fixed intervals of time (the auto-scaling period), after checking one or more metrics defined by the application manager (the tenant). Different scaling metrics can be used, including the average CPU load of the servers, or the number of arriving end-user requests at a time t (R_t). In this work we have chosen R_t. With this number, we compute the application capacity (number of servers K) required to meet the response-time goal agreed between the tenant and the end-users (T_{A-SLA}), computed as follows:

$$Th = \frac{T_{A-SLA}}{T_{AVG}} \tag{1}$$

$$K = \frac{R_t}{Th} \tag{2}$$

In Equation 1, T_{AVG} is the average VM time required to process a request. This is a current value, measured since the last auto-scaling event. Thus, Th is the number of requests that a VM is able to serve obeying the A-SLA. Equation 2 then computes the number of VMs required to deal with request rate R_t. This capacity management scheme is similar to the one used in [15]. In that work they used the maximum number of requests that a server is able to manage without violating the T_{A-SLA}. However, as the use of that value requires *a priori* knowledge of the capacity of servers, we have replaced it with Th, which can be computed dynamically as the execution goes.

Auto-scaling policies can be divided into two broad groups: *reactive* and *proactive*. Policies in the first group make decisions based on the current value of the scaling metrics. In contrast, proactive policies use estimations of future values of the metrics, making auto-scaling decisions with these estimations. This is done because there is a lag from the time a new resource is added/removed until the effects of this action are perceptible, and proactive policies aim to have resources ready before they are needed. Once a decision is made, the auto-scaler spends a cool-down period in which resource changes are not allowed, to avoid oscillation issues. After this time, the results expected from the previous decision can be checked and, maybe, reconsidered.

Although many complex auto-scaling policies have been proposed and developed [19], we have selected two simple ones (one reactive, one proactive) that use metric R_t:

- **Reactive (RT)**: This policy [15] manages resources adapting them to the *current* request rate. It computes the number of required servers using

equations 1 and 2 and then uses the management interface to add or remove servers. The main drawback of this strategy is that its performance depends strongly on the T_D and the T_R of the VMs, as the decisions will not be implemented immediately. Note, though, that if the management times and the period between scaling decisions were zero, this would be an optimal strategy.

- **Moving Window Average (MWA)**: This proactive policy [15] operates like the previous one but, instead of using the current value of R_t, it averages the request rate during a past window of some duration to predict the rate at a *future* point in time. The idea behind this policy is that at time t we can have an estimate of the request rate at time $t + T_D$, making an educated guess about the usefulness of resources not at the current time, but when they are already deployed. As the window size is a parameter with a high impact on the performance of MWA, and it depends on specifics of each workload, we performed a sensitivity analysis using several values for this parameter. The best results were obtained using a window size of twice the deployment time ($2 \times T_D$) and no further improvements were achieved increasing this size.

When scaling-out, new resources are requested from the cloud provider (unless there are enough VMs in the backlog stage ready to be recycled). When scaling-in, it is also necessary to determine which particular VMs will be removed from the application. This could be done using different strategies, such as choosing the *less loaded* VMs or the *last added* ones. Although the selection of this strategy has an impact on the auto-scaling process, for the sake of simplicity we have chosen for this work the *last added* strategy.

4 Experimental setup

In this section we introduce the simulation-based framework used to evaluate auto-scaling policies under different values of VM management times, T_D and T_R. After describing the input workload used to feed the simulator, we present the set of experiments carried out, designed to provide meaningful answers to the three issues outlined in Section 2.3.

4.1 Input workload

The workload contains the sequence of requests that end-users submit to a given application. In this work we use an actual workload captured from the production web server of the University of the Basque Country UPV/EHU [10]. This workload follows a daily periodic pattern receiving, on average, more requests on workdays than during the weekend. We have focused on the load generated during one particular, representative work day, depicted in Figure 3. The figure shows the number of requests per minute that arrive to the web server.

In this time-scale (minutes) we can set apart four periods following different patterns. The first ($P1$) and fourth ($P4$) periods show a decreasing behavior in which the number of requests is lower as time passes; the decreasing rate is higher in $P4$. The second period ($P2$) is of intense growth, while the third one ($P3$) is neither of growing nor of decreasing demand, but shows very strong changes in very short periods of time.

Other works on auto-scaling use specific workload patterns of short duration. However, actual workloads do not follow a single pattern, because the dynamics of users change with time. The use of actual workloads allows us to reach more general conclusions about the behavior of auto-scaling policies.

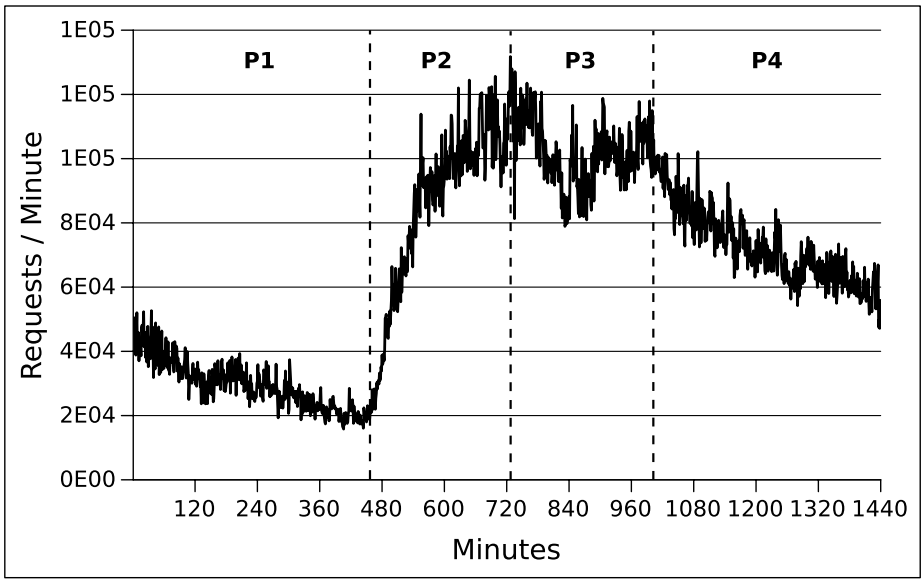


Figure 3: Representation of the real workload extracted from the web server of the University of the Basque Country UPV/EHU. The duration is one day and the plot represents the number of requests per minute.

A main limitation of actual workloads is that they are "as is", a record of an actual behavior. However, it is possible to derive realistic (but not real) workloads from them, varying parameters such as the request rate (e.g. halving or doubling it) or the per-request computational cost. In particular, we have simulated request processing times by sampling a uniform distribution in the interval [50, 250] ms, instead of using the actual ones.

4.2 Simulation environment

We have developed a custom, event-driven cloud simulator that allows us to model and evaluate the dynamics of an auto-scaling environment in which an application can acquire and release resources (VMs) based on the decisions

made by an auto-scaler. The simulator implements several auto-scaling policies, among them RT and MWA. It is also able to capture many performance metrics from a hosted application and the infrastructure. In particular, we have used the following metrics in our evaluations:

- VM hours: Given an application, this metric summarizes the total number of hours that the assigned VMs spend turned on (including the deployment and shutting down time). This metric is related to the utilization cost to be paid by the tenant to the cloud provider.
- Number of VMs: Given an application, this metric summarizes the number of used VMs at a given time. In this work we use it to graphically represent the amount of resources reserved by an application as the execution progresses. Notice that this metric by itself is not a cost indicator; we have to also consider the time that the VMs spend turned on (the VM hours metric explained above).
- Response time: This metric measures the service time of requests submitted by end-users. Sometimes this metric is presented as a simple average, but this is not a good indicator of the number of A-SLA violations. Instead, we will use the percentile 95 of the response time (T_{95}), that is, the value below which 95% of the response times are. When T_{95} is below the A-SLA, we say that the system guarantees a percentile 95 goal of the A-SLA. For this work we have chosen a 1000 ms value for the A-SLA. This choice is motivated by the value used in [15] that was based on recent studies [13] [18] [23] [30] that indicate that 95 percentile guarantees of hundreds of milliseconds are typical.

4.3 Design of the experiments

We have carried out an extensive experimentation with a total of 160 different simulation runs, gathering in each run the metrics detailed above. The parameters varied in the different experiments were:

- We tested the two auto-scaling policies described previously: RT and MWA.
- The scaling-related metrics that trigger scaling actions can be checked with different granularity. We have considered auto-scaling periods of 1, 5, 10 and 60 seconds.
- We have tested five different values of T_D: 1, 5, 10, 60 and 300 seconds. The last value is close to those reported for commercial cloud providers [21].
- For T_R we have tested four values: 1, 5, 10 and 60 seconds.

After a scaling decision is made, a cool-down period is applied (T_{CD}) to allow the resource change to be noticeable in terms of performance. This has to be, at least, the deployment time (to have the VM ready) plus some extra time to have the VM working fully, that is, serving end-user requests. In this work we have set it as twice the deployment time ($T_{CD} = 2 \times T_D$).

Considering all these parameters, we define an *auto-scaling configuration* as a tuple $(P/F/T_D/T_R)$ where P is the auto-scaling policy, F is the auto-scaling period and T_D and T_R are the deployment and release times respectively. For example, the following auto-scaling configuration

$$MWA/10/60/1$$

represents the use of the MWA auto-scaling policy, an auto-scaling period of 10 seconds, a deployment time of 60 seconds and a release time of 1 second.

As the time required by a VM to process a request is generated randomly (see Section 4.1), we have repeated each experiment 10 times using a different seed each time. Thus, the numerical results reported in the following sections are averages of 10 simulation runs.

5 Cost boundaries and baselines

The purpose of this section is to set up some performance boundaries and baselines allowing us to put in context the merits of the different configurations described in the previous section. First, we compute the "perfect" number of VMs that an application would require to meet the A-SLA (for *all* the end-user requests) with and without the use of an auto-scaler. For this, we assume total knowledge of the incoming workload and zero management times, in such a way that resources are ready when the are needed and only while they are needed. This is an utterly unrealistic scenario, but it is useful to set up a lower bound on resource requirements. We also carry out similar computations in more realistic scenarios involving the use of auto-scalers and typical management times (similar to those offered by actual cloud providers). The obtained results have been depicted in Figure 4. Next we analyze these results in detail.

5.1 Ideal boundaries with and without an auto-scaler

The first cost boundary that we want to establish corresponds to a perfect auto-scaling scenario. Let us assume that management times are zero (requests to the cloud provider are implemented instantaneously). Let us further assume that the auto-scaler has continuously updated information about the current number of VMs, their utilization, the incoming requests and the time required to process them (the scaling metric). Under these ideal circumstances, the auto-scaler can compute with total precision the number of active VMs needed to guarantee that the percentile 100 of the response time is within the A-SLA. This configuration results in a perfect assignment of resources: VMs are reserved only

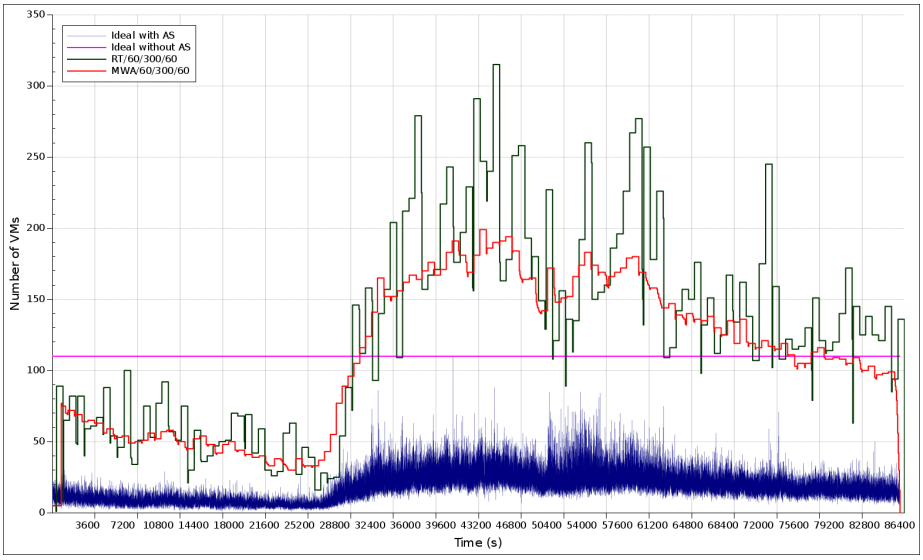


Figure 4: Number of VMs per second required by an application to serve the end-user requests meeting the A-SLA. The blue line corresponds to an elastic application with a perfect auto-scaler. The pink horizontal line corresponds to a non-elastic application with the maximum number of VM/s of the previous configuration. The green and red lines correspond to realistic configurations RT/60/300/60 and MWA/60/300/60.

when they are necessary, for the necessary period, and with full utilization; see the blue line in Figure 4.

To implement this assignment of resources, the auto-scaler needs to know *a priori* the computational demands of incoming end-user requests, and can adapt to the workload immediately. This is possible in a simulation environment, but unattainable in reality. Also, zero values for T_D and T_R are unreachable. Using the terminology described in Section 4.3, the blue line would correspond to a RT$\star$/0/0/0 auto-scaling configuration, where RT$\star$ is an auto-scaling policy with perfect knowledge of the workload.

We may wonder what the cost would be for the same application (under the same workload) if implemented in a non-elastic way, i.e. without using an auto-scaler. Again, with perfect knowledge of the workload, we could dimension the application to cope with the maximum peak of demand, guaranteeing that in all cases the A-SLA is met. This policy is called *always on* in [15]. The application will have, since its initial deployment, the necessary number of VMs, depicted in Figure 4 as a horizontal pink line. Note that this number is the peak number of VM/s in the RT$\star$/0/0/0 scenario. In this context, management times are irrelevant, as there are no scaling events.

5.2 Baseline cost with realistic VM management times

In the previous section we have established some ideal cost boundaries. Now we focus on assessing realistic scenarios of elastic applications under unpredictable workloads with VM management times close to those offered by commercial cloud providers, setting a baseline with which to compare the effects of shrinking management times in terms of both cost and performance. If this baseline is close to the RT$\star$ ideal boundary, then no effort would be required to reduce T_D and T_R. However, as we will see, this is not the case.

We have tested two auto-scaling configurations with different auto-scalers:

- A reactive (RT) configuration with an auto-scaling period of 60s, $T_D = 300$ and $T_R = 60$ (RT/60/300/60)
- A proactive (MWA) configuration with the same parameters (MWA/60/300/60)

In both cases, the cool-down time (minimum time between auto-scaling decisions) is $2 \times T_D$. We have verified that both configurations guarantee that T_{95} is below the A-SLA limit.

The results of the corresponding experiments are also represented in Figure 4. The green line represents the number of VM per second required by the reactive configuration. The long management times (that also imply long cool-down times) make this configuration *very far* from the optimal. See the "stepped" shape of the green line representing VM/s, and also the wide oscillations in the number of deployed VMs.

The red line corresponds to the proactive configuration. Its shape is also stepped, because of the long management times, and shows the difficulties that the auto-scaler has to rapidly adapt to the input load. However, this policy results in a better use of resources: the number of VMs required to meet the A-SLA is much lower than with the reactive policy, showing the effectiveness of using predicted values of the scaling metric instead of the current ones. Also, oscillations are greatly reduced: the number of deployed VMs is more stable.

We must insist on stating that, in real scenarios it is impossible to develop the perfect auto-scaler due to many reasons. In particular, we can neither know in advance the exact number of requests that will arrive (even when using predictive techniques), nor the time required to process them. However, the obtained unachievable results can be used to evaluate how far from them other policies perform.

These initial experiments show clearly that, with management times similar to those available nowadays, the proactive auto-scaler performs better than the reactive one. This justifies the extensive body of literature on proactive auto-scaling (see [19] for more details). MWA is capable of partially compensating the delays in resource availability resulting from the VM management times. However, the costs derived from both RT and MWA are still excessive: both policies assign more resources than required to guarantee that response times are within bounds (the A-SLA). If we compare Figure 4 with the shape of the workload (see Figure 3 in Section 4.1), we can see that for the decreasing period

P1, the difference between the ideal and the achieved number of VMs is not very large. However, the gap grows greatly at the beginning of P2: the auto-scaler takes too long to adapt to the increasing number of requests. Both the RT and MWA auto-scalers require more VMs than those assigned by the non-elastic configurations, and this is due to the need for processing, in addition to the incoming requests, the backlog of accepted but not yet finalized requests in the servers.

In the following section we dig further into the performance of auto-scaling configurations, exploring their efficiency when using cloud resources.

6 Analysis of resource utilization

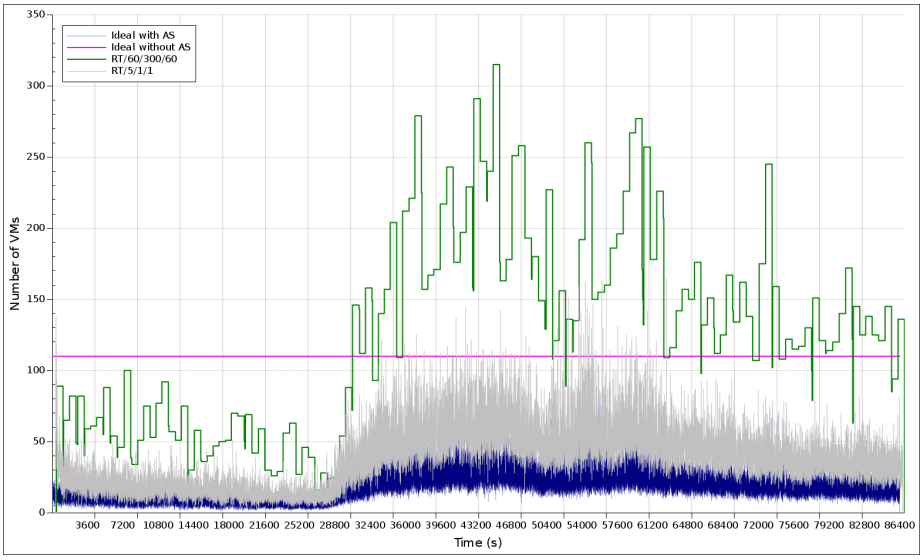
In the previous section we have shown how shortening VM management times results in a reduction in the time required by the application to adapt to the incoming workload. Now, using the presented performance boundaries, we want to measure this effect in terms of reduction in resource utilization.

We have summarized the results of the experiments for some representative configurations in Figure 5. The top graph represents reactive auto-scaling, while the bottom graph is for proactive auto-scaling. For both policies, we have tested a configuration with long management times (close to those currently available from commercial providers), and a configuration with very short management times (not available from current providers).

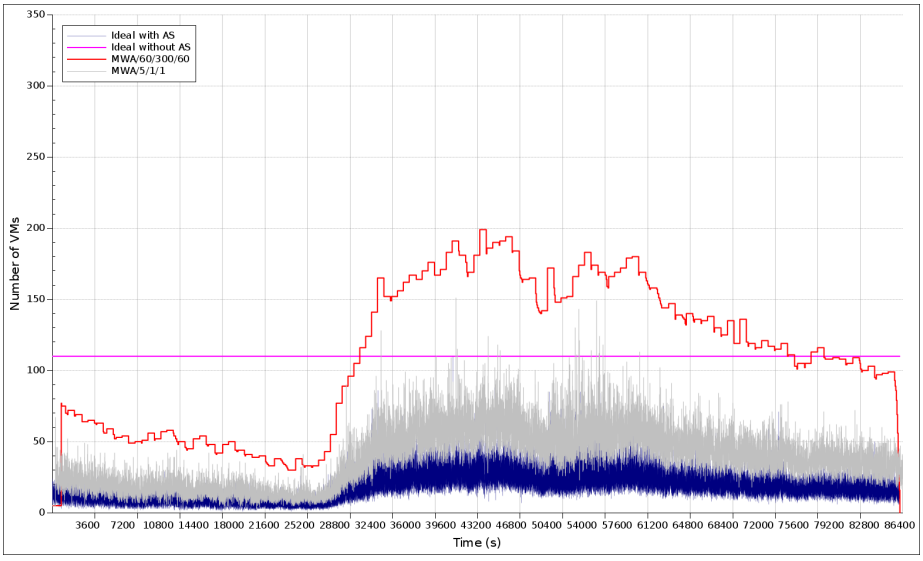
For both policies, short management times (red line) results in *much* improved resource utilization compared to those achieved with "commercial" times – always complying with the A-SLA. These results are not as good as with the perfect auto-scaler, but they are much better than the static option. On average, RT/60/300/60 requires 130.53 VMs, while RT/5/1/1 can cope with the same workload using just 52.16 VMs. Note also how the use of a shorter auto-scaling period results in a significant increase in scaling events, allowing the auto-scaler to adapt to the workload more rapidly. The analysis for the proactive policy is similar: the average number of VMs is reduced from 114.26 to 45.11.

If we compare them with the theoretical boundaries, we can see how */5/1/1 configurations are closer to the RT$\star$/0/0/0 configuration (in relative terms, using commercial management times as a reference), but the use of resources is not optimal. This is to be expected, because the optimal policy has perfect knowledge of the workload, and this knowledge is, obviously, not available to a real auto-scaler. However, we deem this as an excellent result. It it also to be noticed how, with commercial management times, average use of resources exceeds the number of 110 VMs of the non-elastic alternative, meaning that the use of an auto-scaler is actually counterproductive. However, using short management times, auto-scaling shows its merit reducing the average use of resources well below that number.

Figure 5 only shows the (dynamic) behavior of a few configurations, allowing us to extract some conclusions about the way they acquire and release VMs. Now we will focus on a more exhaustive study of other configurations, using in



(a) RT



(b) MWA

Figure 5: Representation of the number of VMs per second required by an application to serve the end-user workload, complying with the A-SLA. Blue: configuration RT⋆/0/0/0. Pink: static configuration (no auto-scaler). Green: RT/60/300/60 (a) and MWA/60/300/60 (b). Red: RT/5/1/1 (a) and MWA/5/1/1 (b).

this case as performance metric the number of VM hours required to process the incoming workload, which is closely related to the cost of the configuration. We have chosen, for the sake of clarity, only a subset of possible auto-scaling periods, but the results have been validated for other configurations. In particular, we will analyze these configurations (where * represents all the possible values for a parameter):

- RT/5/*/* and MWA/5/*/*: configurations with very small auto-scaling period, to be compared with
- RT/60/*/* and MWA/60/*/*: configurations with a "normal" auto-scaling period.

6.1 Average results with reactive auto-scaling

The results for the reactive auto-scaler have been summarized in Figure 6. The columns in the graphs represent the VM hours used by the different configurations: the graphs on the left correspond to a 5 s scaling period, while the ones on the right are for a 60 s scaling period. Columns are arranged in sets. A set corresponds to a value of T_D, while each individual column in a set corresponds to a value of T_R. We can extract the following conclusions:

- The benefits of using short management times can only be obtained with short auto-scaling periods.
- Then, with a frequently-invoked auto-scaler, important cost reductions can be achieved if deployment times are below 10 s.
- However, to take advantage of the benefits of enjoying a short T_D, T_R must be short too.

In summary: the best configurations for the RT reactive policy are those with short management times, but only if the auto-scaler operates with a correspondingly increased frequency. These configurations make the auto-scaler more capable of adapting to the current workload. As an example, the number of VM hours used in a realistic RT/60/300/10 configuration is three times larger than that required in a hypothetical RT/5/10/5 configuration. There are, thus, compelling reasons to ask the cloud provider to implement mechanisms to reduce management times.

6.2 Average results with proactive auto-scaling

Now we will perform the same analysis of averaged cost but using the proactive MWA auto-scaling policy. We want to remark that MWA has not exact knowledge of future requests, only an estimation based on past behavior. Results are depicted in Figure 7, which is arranged as explained for Figure 6.

There are significant differences between both figures (that for RT vs. that for MWA). The first difference is that with MWA the auto-scaling frequency is

not a critical parameter: graphs for 60 s and 5 s auto-scaling periods are almost identical. And the same can be said about the release times: with MWA they do not affect results in a significant way.

The two graphs of Figure 7 still show the advantages of using short deployment times. A value of 10 s is good enough to obtain important benefits. Values over 60 s are too long to allow the predictions provided by MWA to be precise enough, resulting in an excessive allocation of resources – the priority is always to comply with the A-SLA.

In Figure 7 we can see a clear difference between the results for $T_D = 10$ (and below) and $T_D = 60$. To see what happens for values in the middle, we have depicted in Figure 8 the results of MWA/5/*/* configurations with T_D from 10 to 60 with a step of 10. As we can see, the number of VM hours is proportional to T_D for values over 10, but does not decrease significantly for lower values of this management time. The explanation has to be in the prediction ability of the MWA technique: long-term predictions tend to be less accurate than short-term predictions.

In summary, and using the same example of the previous section, the number of VM hours used in a realistic MWA/60/300/10 configuration is 2.5 times larger than that required in any hypothetical MWA/*/10/* configuration. This confirms again that cloud tenants would benefit from I-SLAs specifying short deployment times, if cloud providers were able to implement mechanisms to guarantee them.

7 Dissection of the lifetime of VMs

The previous results have been expressed in terms of VM hours, which translate immediately into costs paid by the tenant. We have seen remarkable reductions in this metric resulting from the availability of short management times – or, the other way around, remarkable overheads resulting from long management times. We now analyze these overheads, to better understand how they can be reduced. We do so by analyzing, together with the number of VMs used in the different configurations explored in the experiments, the lifetime of those VMs, to understand which portion of that lifetime is useful time (active time, T_A) compared to overheads (T_D, T_B, T_R), and to what extent that active time is effectively used. This way, we can understand the relationship between policy-parameters and performance-overheads.

We have summarized in Table 1 different parameters and metrics for a set of configurations; for the sake of clarity we have included only */{5,60}/*/10 scenarios, although similar results were obtained with other configurations. These are average results for the total time required to consume the workload (and for 10 simulation runs). We have included in the table:

- The configuration parameters. We use different values of policy, auto-scaling period and deployment time, but the release time has been fixed to 10 s.

- T_{VM}: The cost in terms of VM minutes. This can be achieved with a few VMs deployed for a long time, or with many short-lived VMs.
- A dissection of the utilization of time by the VMs, indicating how much of this time is deployment time (TT_D), active time (TT_A), backlog time (TT_B) and release time (TT_R). All times are expressed in minutes, and are aggregates of all the VMs used.
- The T_{95} time of each configuration. Notice that only in one instance (MWA/5/300/10) this time (2646 ms) does not comply with the A-SLA (1000 ms).
- The AS Events column summarizes the number of scale-in decisions made by the auto-scaler. As the experiments collect all the lifetime of the applications, this number is also the number of scale-out decisions. A single scale-in (scale-out) decision may result in the addition (release) of multiple VMs.

As the release time is 10 s (1/6 m), values of column TT_R (in minutes) divided by 1/6 (multiplied by 6) indicate the number of VMs that have been *actually* released, while column AS Events indicates the number of times that the auto-scaler has decided to remove VMs. Remember that each scaling decision may affect adding / removing several VMs, and that the adding process could be accelerating if VMs are ready for recycling.

Let us pay attention first to the upper part, corresponding to the RT reactive policy. We confirm with this table the conclusions extracted from Figure 6: the best configurations are those with short deployment times but only if the scaling period is reduced accordingly. Therefore, the best configuration is RT/5/1/10. Focusing on this, we can observe that most of the time spent by the VMs is active time; deployment and release overheads are relatively low. These overheads reflect that the configuration is *very* reactive: a TT_R=10441.10 m means that 62646 VMs have been turned on and then off during the life of the application. Less reactive configurations, such as RT/60/300/10 shows 5244 changes in the VM pool. In both cases the collection of active VMs is able to cope with the workload without exceeding the A-SLA, but the first scenario uses on average just 52.16 VMs with each VM turned on for 1253.03 minutes, while the other requires 130.53 VMs with each VM turned on on average for 1391.62 minutes (these values are not in the table, they have been extracted from the simulation logs).

Also, we can see that backlog times (TT_B in the table) in RT/5/1/10 are relatively high, meaning that when the auto-scaler decides to remove a VM, it needs a non-trivial time to complete the pending requests. This also means that when a new VM is to be added, there is a high probability of recycling one in the backlog phase, thus avoiding a release and a deployment period. In contrast, configurations with long scaling periods and/or long deployment times cannot really take advantage of the opportunity of recycling VMs. Furthermore, a long TT_B is an indicator of a better utilization of the reserved resources, as it implies

Table 1: Parameters and metrics captured for the following configurations: RT/{5,60}/*/10 and MWA/{5,60}/*/10. The values correspond to the average of 10 different simulation runs.

				T_{VM}					
Policy	AS Period	T_D	T_{VM}	TT_D	TT_A	TT_B	TT_R	T_{95}	
RT	5	1	65358.99	1044.12	53677.56	196.21	10441.10	951.53	
		5	68534.08	4243.86	55652.35	150.31	8487.56	934.62	
		10	78109.35	7014.30	63977.17	103.75	7014.13	894.17	
		60	155052.69	20130.40	131558.70	8.69	3354.90	495.06	
		300	181648.67	27790.50	152931.76	0.23	926.18	434.53	
	60	1	116572.10	322.56	113008.02	16.07	3225.45	913.39	
		5	122244.39	1681.35	117186.36	14.15	3362.53	833.95	
		10	130431.22	3571.08	123277.95	11.28	3570.91	798.09	
		60	169342.18	16924.00	149597.40	0.28	2820.50	434.45	
		300	182837.57	26234.50	155728.60	0.16	874.31	364.65	
MWA	5	1	58497.08	745.94	50115.51	176.32	7459.31	580.57	
		5	56773.61	1777.49	51332.82	108.50	3554.81	576.39	
		10	57634.66	1940.23	53695.10	59.27	1940.06	544.64	
		60	141626.71	3152.70	137944.41	4.32	525.28	410.27	
		300	160187.77	3215.00	156865.25	0.52	107.00	2646.68	
	60	1	55393.81	123.35	54008.46	28.64	1233.36	865.12	
		5	54561.55	389.85	53368.43	23.74	779.53	809.35	
		10	55971.86	636.01	54680.85	19.15	635.85	789.34	
		60	141652.46	2908.20	138255.80	3.93	484.53	432.73	
		300	159543.48	3095.00	156345.00	0.48	103.00	366.02	

that active MVs, including those marked to be removed, are busy processing requests. Note how, for costly configurations (those with high T_{VM}) TT_B is close to zero, meaning that resources are underutilized.

The proactive policy MWA is always better than RT: for comparable configurations, cost is lower (see the T_{VM} column), T_{95} is shorter (except for an invalid configuration, MWA/5/300/10) and the number of scaling events is much smaller. These good results are equally good for any value of $T_D <= 10$, and even when the auto-scaling period is 1 min.

Focusing on the user experience, we consider that any configuration complying with the A-SLA is valid. Although shorter response times are not a requirement, they will be perceived positively by users. In other words: no extra resources should be added to reduce the T_{95} when it is just below the A-SLA limits, but when comparing complying configurations with similar cost, the one with smaller T_{95} is the best. Note how proactive configurations using

fewer resources than their reactive counterparts achieve, additionally, shorter T_{95} values. This also means that there is room for improvement in terms of cost, as deployed VMs are not used fully.

We want to remark that MV minutes cannot be translated directly into cost when taking into consideration the diverse payment models used by commercial cloud providers. For example, with Amazon EC2, a user pays for a full hour once she has a new MV allocated; with Windows Azure, the user pays for the first ten minutes and, after that, billing goes by the (active) minute. This means that when too many AS events are executed, the application will not take full advantage of a pre-paid time slot (1 hour or 10 m). The capability of recycling machines reduces this effect. However, the *pay-as-you-go* concept should be enhanced to adjust better to the actual use of resources. Also, auto-scalers could adapt the scaling decisions to match those pre-paid time slots, but this may result in bad performance levels or exceedingly high costs (under- or over-provisioning) due to the reduced capacity to adapt to the input load.

8 Related work

In this section we discuss some previous works on resource management for elastic applications in cloud environments. To the best of our knowledge, only a few works have measured the VMs management times employed by commercial providers. However, none of those studies have been devoted to deeply analyzing the way these times affect the capability of an application to meet the A-SLA or the effectiveness of the auto-scaling policies. All the works have chiefly focused on developing mechanisms to actually reduce those times, mainly the deployment time.

In [21], the authors measured the time employed by three real-world cloud providers (Amazon EC2 [2], Windows Azure [11] and Rackspace [9]) to deploy VMs. In particular, they analyze the relationship between the deployment time and other factors such as the time of the day, the image size, the instance type and the number of instances acquired at the same time. They argue that this information is important to time-critical applications that rely on the elastic resources provided by cloud infrastructures and that it is crucial to allow the auto-scaling mechanisms to make correct decisions [20] [22]. In [26] a more detailed analysis of deployment times is carried out, which takes into account not only the image size but also additional content required for server startup downloaded from the Amazon cloud infrastructure. Based on their findings, they developed a mechanism to reduce that content and the size of the VM images, which results in up to four times reduction of the storage size, and also three times reduction of the VM startup time.

Instead of focusing of reducing data transfers using data compression, other works aim to reduce management times via capacity management policies. For instance, the *AutoScale* system described in [15] claims to reduce significantly the number of servers required by an application in a data center by of increasing the shutdown time of the VMs (what we call backlog time) in order to increase

the probability of reusing VMs set to be shutdown. In [16] authors focused on reducing the resume time of the VMs, that is, that of redeploying a previously suspended VM. They implemented a real system using the open-source virtual machine emulator Qemu [8], and were able to reduce the resume time to about 3 s, less than 1/10 of the default implementation. These results resulted in the development of Dreamserver [17], an architecture to deploy virtualized services just-in-time, such as web applications that are suspended when idle, and resurrected in less than one second when the next request arrives.

Razavi et al. study both approaches in combination. They dealt with transferring the VM image from the storage node to the host in [25] and presented Squirrel, a system that stores all VM images of a data center on all physical servers. Then, in [27], the actual booting process of the VM is studied, proposing to pre-boot a VM image in advance and take a snapshot with minimal hardware resources that can be resumed very fast (2-3 s on average).

Regarding the behavior of current auto-scalers, they mainly use reactive techniques by means of rules set by the owners of the application. Tenants choose a particular metric, such average CPU time, request count or inbound/outbound traffic, and configure thresholds (or timetables) to trigger specific scaling-out or scaling-in actions. In addition, cool-down periods must be configured in order to prevent the auto-scaler from collecting information while the instance is initializing, because during that time the collected usage metrics would not be reliable and could cause undesirable oscillation effects. The most prominent examples of available auto-scalers are AWS Auto Scaling [1], Azure Autoscale [4], Compute Engine Autoscaler [6], Bluemix Auto-scaling [5] and OpenStack SenLin [7]. It is worth noting that AWS Auto Scaling has introduced recently *Target Tracking* [3], which automatically scales applications after selecting only a value for a target metric and, if desired, the cool-down period. Although Amazon does not provide specifics about the internal behavior of this auto-scaler, it claims to be proactive by constantly monitoring and making predictions of the target metric in order to automatically scale the applications. The most used prediction techniques that could be used for proactive auto-scalers, such as control theory or machine learning, can be found in [28].

9 Conclusions and future work

Throughout this work we have discussed in depth the effects that the reduction of VM management times (mainly deployment and release times) has on the cost and performance of elastic applications deployed in the cloud. We have shown how short management times can result in using 1/3rd of the resources required with current management times, always with the strict restriction of complying with the A-SLA. Although other works have addressed the reduction of the VM deployment time, none of them has deeply analyzed quantitatively the benefits of this reduction.

If the auto-scaler is reactive, performance benefits can only be achieved using proportionally shorter scaling periods. Proactive auto-scalers do a good

job predicting future demands, and can adjust more smoothly to the incoming workload without excessive modifications in the resource pool. We can state that, with short management times, reactive auto-scalers can adjust better resources to the workload, but adding a simple proactive policy allows going a step further in terms of resource savings.

In conclusion, auto-scaling elastic applications *need* a reduction of current VM (or container) management times. This is mostly beyond the control of the application: the infrastructure manager must implement strategies to shorten deployment/release times. These strategies can be implemented using pre-deployed, "dozing" VMs that will not be used until they are required by the application (scaling-out event). There would be hidden costs, as some reserved but idle resources must be kept. However, using appropriate technical and economic policies, this cost could be very small, and more than counterbalanced by the derived benefit. Analyzing this tradeoff is part of our future work in this area. In particular, we are interested in developing strategies to be implemented by the cloud infrastructure managers. The goal is to reduce the management times in a way that is beneficial for both tenants and infrastructure owners. Applications could take advantage of cost reductions derived from the better utilization of resources (without putting A-SLA compliance at risk), while infrastructure managers will use fewer resources to provide high-quality services to the tenants, thus reducing the energy consumption or hosting more applications with the same amount of resources.

Acknowledgments

This work has been partially supported by TIN2016-78365R (Ministry of Science and Technology) and the Research Groups 2013-2018 (IT-609-13) program. Jose A. Lozano is also supported by BERC program 2014-2017 (Basque government) and Severo Ochoa Program SEV-2013-0323 (Spanish Ministry of Economy and Competitiveness). Jose Miguel-Alonso is member of the HiPEAC European Network of Excellence. We want to thank the technicians of the IT department (CIDIR) of the University of the Basque Country (UPV/EHU) for gathering and providing the workloads used in this study.

References

[1] (2018) Amazon AWS Auto Scaler. `https://aws.amazon.com/autoscaling/`, [Online; accessed 23-April-2018]

[2] (2018) Amazon EC2. `https://aws.amazon.com/ec2/`, [Online; accessed 23-April-2018]

[3] (2018) Amazon Fast Tracking. `https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html`, [Online; accessed 23-April-2018]

[4] (2018) Azure Autoscale. https://azure.microsoft.com/en-us/features/autoscale/, [Online; accessed 23-April-2018]

[5] (2018) Bluemix Auto-scaling. https://www.ibm.com/developerworks/cloud/library/cl-bluemix-autoscale/, [Online; accessed 23-April-2018]

[6] (2018) Compute Engine Autoscaler. https://cloud.google.com/compute/docs/autoscaler/, [Online; accessed 23-April-2018]

[7] (2018) OpenStack SenLin. https://wiki.openstack.org/wiki/Senlin/, [Online; accessed 23-April-2018]

[8] (2018) Qemu. www.qemu.org/, [Online; accessed 23-April-2018]

[9] (2018) Rackspace. www.rackspace.com/, [Online; accessed 23-April-2018]

[10] (2018) University of the Basque Country UPV/EHU. www.ehu.eus/, [Online; accessed 23-April-2018]

[11] (2018) Windows Azure. https://azure.microsoft.com/, [Online; accessed 23-April-2018]

[12] Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P (2017) Elasticity in cloud computing: State of the art and research challenges. IEEE Transactions on Services Computing PP(99):1–1, DOI 10.1109/TSC.2017.2711009

[13] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key-value store. In: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, ACM, New York, NY, USA, SOSP '07, pp 205–220, DOI 10.1145/1294261.1294281, URL http://doi.acm.org/10.1145/1294261.1294281

[14] Galante G, Erpen De Bona LC, Mury AR, Schulze B, da Rosa Righi R (2016) An analysis of public clouds elasticity in the execution of scientific applications: a survey. Journal of Grid Computing 14(2):193–216, DOI 10.1007/s10723-016-9361-3, URL https://doi.org/10.1007/s10723-016-9361-3

[15] Gandhi A, Harchol-Balter M, Raghunathan R, Kozuch MA (2012) Autoscale: Dynamic, robust capacity management for multi-tier data centers. ACM Trans Comput Syst 30(4):14:1–14:26, DOI 10.1145/2382553.2382556, URL http://doi.acm.org/10.1145/2382553.2382556

[16] Knauth T, Fetzer C (2013) Fast virtual machine resume for agile cloud services. In: Cloud and Green Computing (CGC), 2013 Third International Conference on, pp 127–134, DOI 10.1109/CGC.2013.27

[17] Knauth T, Fetzer C (2014) Dreamserver: Truly on-demand cloud services. In: Proceedings of International Conference on Systems and Storage, ACM, New York, NY, USA, SYSTOR 2014, pp 9:1–9:11, DOI 10.1145/2611354.2611362, URL http://doi.acm.org/10.1145/2611354.2611362

[18] Krioukov A, Mohan P, Alspaugh S, Keys L, Culler D, Katz RH (2010) Napsac: Design and implementation of a power-proportional web cluster. In: Proceedings of the First ACM SIGCOMM Workshop on Green Networking, ACM, New York, NY, USA, Green Networking '10, pp 15–22, DOI 10.1145/1851290.1851294, URL http://doi.acm.org/10.1145/1851290.1851294

[19] Lorido-Botran T, Miguel-Alonso J, Lozano J (2014) A review of auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing 12(4):559–592, DOI 10.1007/s10723-014-9314-7, URL http://dx.doi.org/10.1007/s10723-014-9314-7

[20] Mao M, Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, pp 1–12

[21] Mao M, Humphrey M (2012) A performance study on the vm startup time in the cloud. In: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, IEEE Computer Society, Washington, DC, USA, CLOUD '12, pp 423–430, DOI 10.1109/CLOUD.2012.103, URL http://dx.doi.org/10.1109/CLOUD.2012.103

[22] Mao M, Li J, Humphrey M (2010) Cloud auto-scaling with deadline and budget constraints. In: Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on, pp 41–48, DOI 10.1109/GRID.2010.5697966

[23] Meisner D, Sadler CM, Barroso LA, Weber WD, Wenisch TF (2011) Power management of online data-intensive services. In: Proceedings of the 38th Annual International Symposium on Computer Architecture, ACM, New York, NY, USA, ISCA '11, pp 319–330, DOI 10.1145/2000064.2000103, URL http://doi.acm.org/10.1145/2000064.2000103

[24] Pascual JA, Lorido-Botran T, Miguel-Alonso J, Lozano JA (2015) Towards a greener cloud infrastructure management using optimized placement policies. J Grid Comput 13(3):375–389, DOI 10.1007/s10723-014-9312-9, URL https://doi.org/10.1007/s10723-014-9312-9

[25] Razavi K, Ion A, Kielmann T (2014) Squirrel: Scatter hoarding vm image contents on iaas compute nodes. In: Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, ACM, New York, NY, USA, HPDC '14, pp 265–278, DOI 10.1145/2600212.2600221, URL http://doi.acm.org/10.1145/2600212.2600221

[26] Razavi K, Razorea L, Kielmann T (2014) Reducing vm startup time and storage costs by vm image content consolidation. In: an Mey D, Alexander M, Bientinesi P, Cannataro M, Clauss C, Costan A, Kecskemeti G, Morin C, Ricci L, Sahuquillo J, Schulz M, Scarano V, Scott S, Weidendorfer J (eds) Euro-Par 2013: Parallel Processing Workshops, Lecture Notes in Computer Science, vol 8374, Springer Berlin Heidelberg, pp 75–84, DOI 10.1007/978-3-642-54420-0_8, URL http://dx.doi.org/10.1007/978-3-642-54420-0_8

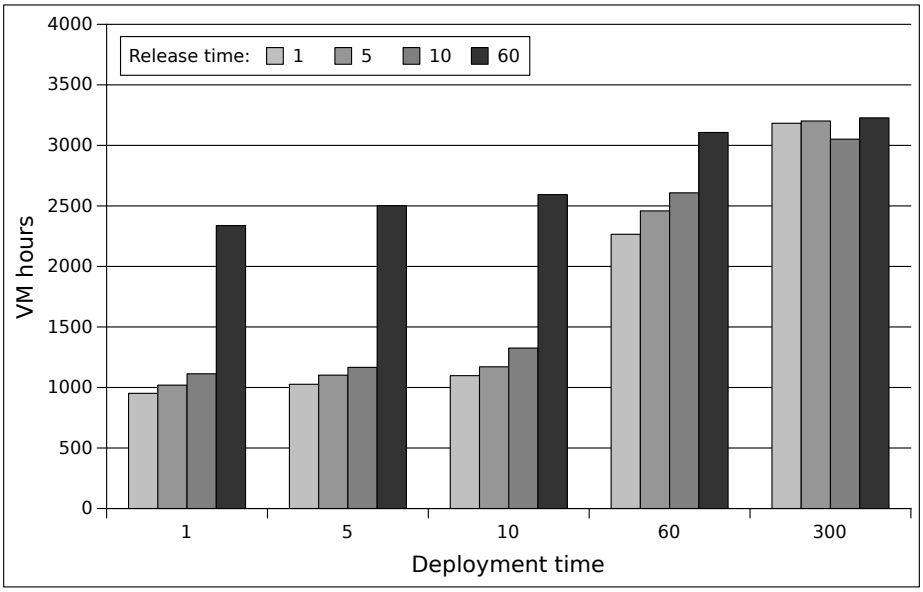
[27] Razavi K, Van Der Kolk G, Kielmann T (2015) Prebaked vms: Scalable, instant vm startup for iaas clouds. In: Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on, pp 245–255, DOI 10.1109/ICDCS.2015.33

[28] T Chen RB, Yao X (2019) A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. ACM Computing Surveys 51(3), DOI 10.1145/3190507

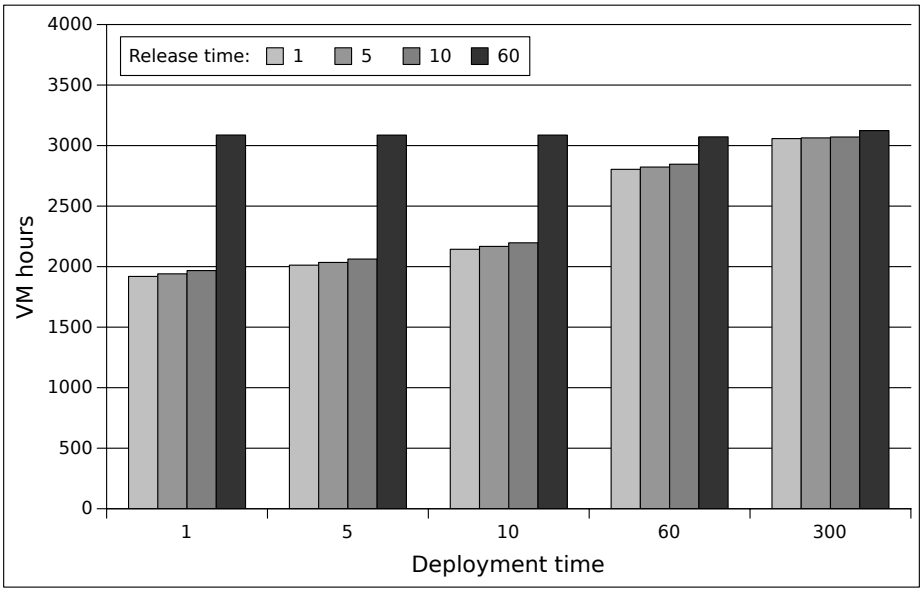
[29] Tighe M, Bauer M (2017) Topology and application aware dynamic vm management in the cloud. Journal of Grid Computing 15(2):273–294

[30] Urgaonkar B, Pacifici G, Shenoy P, Spreitzer M, Tantawi A (2005) An analytical model for multi-tier internet services and its applications. In: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, ACM, New York, NY, USA, SIGMETRICS '05, pp 291–302, DOI 10.1145/1064212.1064252, URL http://doi.acm.org/10.1145/1064212.1064252

[31] Vaquero LM, Rodero-Merino L, Buyya R (2011) Dynamically scaling applications in the cloud. SIGCOMM Comput Commun Rev 41(1):45–52, DOI 10.1145/1925861.1925869, URL http://doi.acm.org/10.1145/1925861.1925869

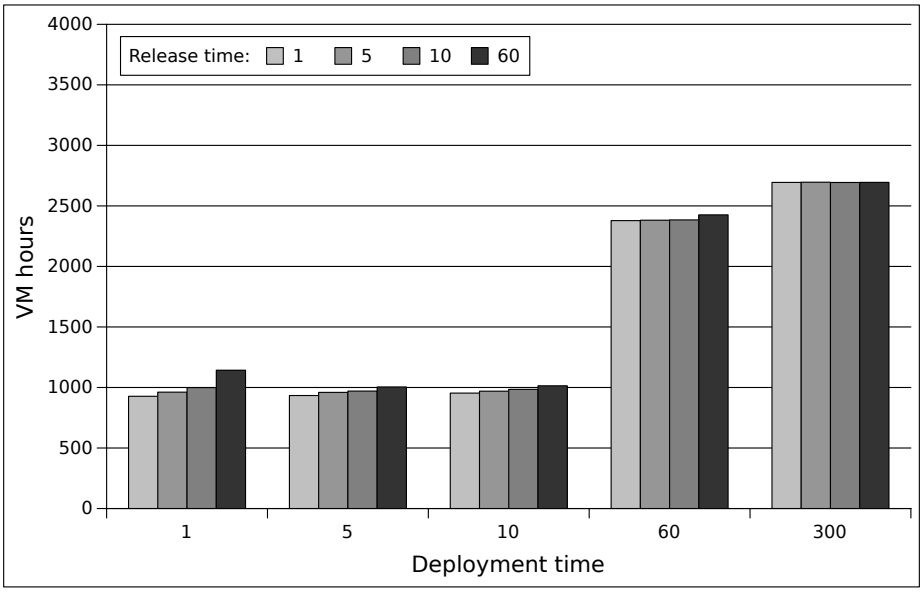


(a) 5 second auto-scaling period.

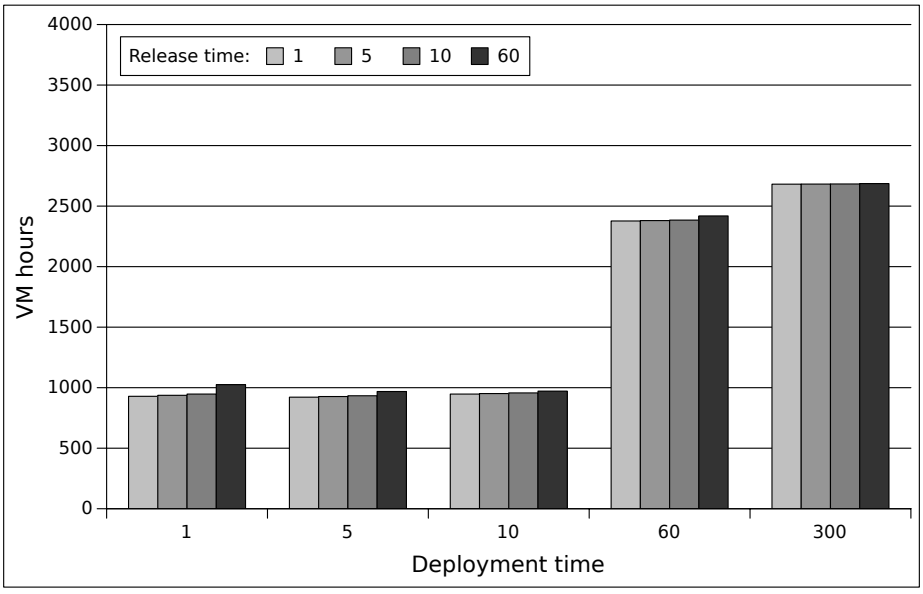


(b) 60 second auto-scaling period.

Figure 6: VM hours used by the application for different configurations of deployment and shutdown times using a reactive auto-scaling policy (RT).



(a) 5 second auto-scaling period.



(b) 60 second auto-scaling period.

Figure 7: VM hours used by the application for different configurations of deployment and shutdown times using a proactive auto-scaling policy (MWA).

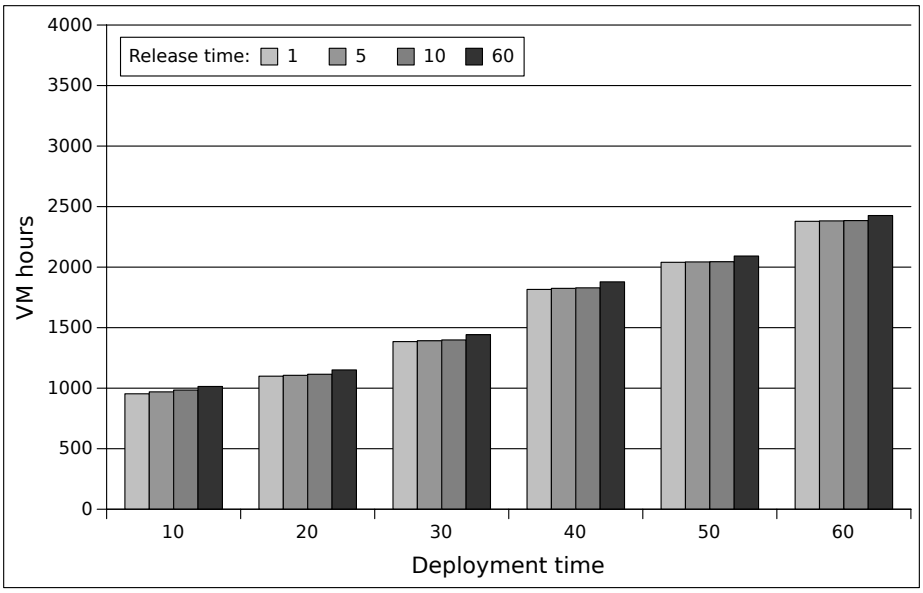


Figure 8: VM hours used by the application for MWA/5/{10,20,30,40,50,60}/* configurations.