Analysing Scalability Strategies for Service Choreographies on Cloud Environments

RAPHAEL GOMES, FÁBIO COSTA and RICARDO ROCHA, Universidade Federal de Goiás

Scalability is one of the major advantages brought by cloud computing environments. This advantage can be even more evident when considering the composition of services through choreographies. However, when dealing with applications that have quality of service concerns scalability needs to be performed in an efficient way considering both horizontal scaling - adding new virtual machines with additional resources, and vertical scaling - adding/removing resources from existing virtual machines. By efficiency we mean that non-functional properties must be offered in the choreographies while is made effective/improved resource usage. This paper discusses scalability strategies to enact service choreographies using cloud resources. We present efforts at the state of the art technology and an analysis of the outcomes in adopting different strategies of resource scaling. We also present experiments using a modified version of CloudSim to demonstrate the effectiveness of these strategies in terms of resource usage and the non-functional properties of choreographies.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Cloud Computing*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids—*Scalability*

General Terms: Cloud Computing

Additional Key Words and Phrases: Cloud Computing, Scalability, Choreography, Auto-Scaling

ACM Reference Format:

Raphael Gomes, Fábio Costa and Ricardo Rocha. 2014. Analysing Scalability Strategies for Service Choreographies on Cloud Environments. *jn* 1, 1, Article 10 (July 2014), 12 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The provision of quality of service (QoS) is one of the main challenges in cloud computing [Blair et al. 2011], since this paradigm must provide assurances that go beyond the typical maintenance activities and must provide high reliability, scalability and autonomous behavior. Many QoS aspects of an application are related with the scalability provided by the hardware resources used to deploy it. As a matter of fact, cloud environments are increasingly used due to their elasticity and the illusion of infinite resources. Increasing degrees of scalability are achieved through the automated management

This work is supported by the Brazilian foundation FAPEG grants #04/2011, #12/2012 and #03/2013.
Author's address: Raphael Gomes, Instituto Federal de Goiás, Goiânia/GO, Brazil; phone: +55 (62) 3227-2700, email: raphael.gomes@ifg.edu.br; Fábio Costa, Instituto de Informática, Universidade Federal de Goiás, Goiânia/GO, Brazil; phone: +55 (62) 3521-1181, email: fmc@inf.ufg.br; Ricardo Rocha, Instituto de Informática, Universidade Federal de Goiás, Goiânia/GO, Brazil; phone: +55 (62) 3521-1181, email: ricardo@inf.ufg.br.

DOI: http://dx.doi.org/10.1145/0000000.0000000

of resources, typically using horizontal scaling, which means changing the amount of resources used, by adding or removing virtual machines (VM) according to policies related to the use of such resources or non-functional properties of the application.

Another strategy for scalability is the use of vertical scaling, i.e. on-the-fly changing of the amount of resources allocated to an already running VM instance, for example, allocating more physical CPU time to a running virtual machine. In a complementary manner, we can have a hybrid approach where we increase both the number and the configuration of virtual machines.

Although there are different scalability strategies they must be used in an efficient way, regarding the consumption of resources. This is due to the fact that a poor management of resources can result in unnecessary spending in the case of public clouds, as well as problems related to energy consumption and loss of investment in the case of private clouds. Another issue that must be taken into account is the QoS offered to applications since some functionalities may not be useful if certain non-functional attributes are not guaranteed [Chung and do Prado Leite 2009].

These challenges are even more evident in the so called Future Internet, which results from the evolution of the current Internet, in combination with the Internet of Content [Daras et al. 2009], the Internet of Services [Papadimitriou et al. 2009] and the Internet of Things [Atzori et al. 2010]. In this new paradigm there are thousands of services belonging to different organizations that have to cooperate with each other in a distributed and large scale environment. This integrated view of services highlighted some problems that were not readily apparent in previous integration efforts, which hardly reached the scale that systems of web services now have [Vincent et al. 2010].

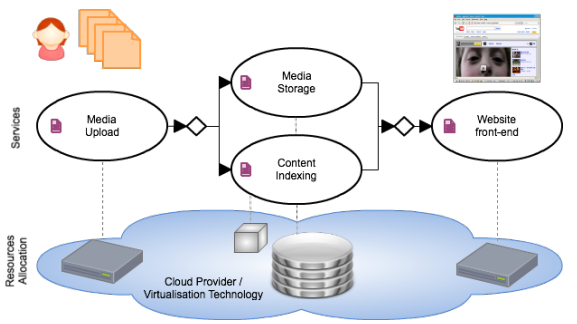
Keeping centralized coordinators for these new types of applications is unfeasible due to requirements like fault tolerance, availability, heterogeneity and adaptability. For this reason, the most promising solution may be the organization of decentralized and distributed services through choreographies. Choreographies are service compositions that implement distributed business processes, typically between organizations in order to reduce the number of exchanged messages and distribute business logic, without the need for centralized coordinators, since each service "knows" when to perform its operations and which other services it must interact with [Barker et al. 2009].

This paper discuss the state of the art in providing scalability for cloud-based service choreographies, considering both technologies and cloud providers. We discuss the outcomes of using cloud environments and the main advantages and disadvantages of adopting different scalability strategies to enact choreographies on cloud resources. We strengthen this discussion with some preliminary evaluation of these strategies. Although this article does not address aspects related to the implementation of choreographies, the analysis presented here can be used as input to different approaches regarding choreography execution in cloud, as well as general applications. The remaining of this paper is as follows: Section 2 presents a motivating example; Section 3 discusses how actual virtualization technologies and cloud providers handle scalability strategies, while Section 4 presents some results of the evaluation of these strategies in choreography enactment. Finally, Section 5 discusses related work and Section 6 presents the final remarks.

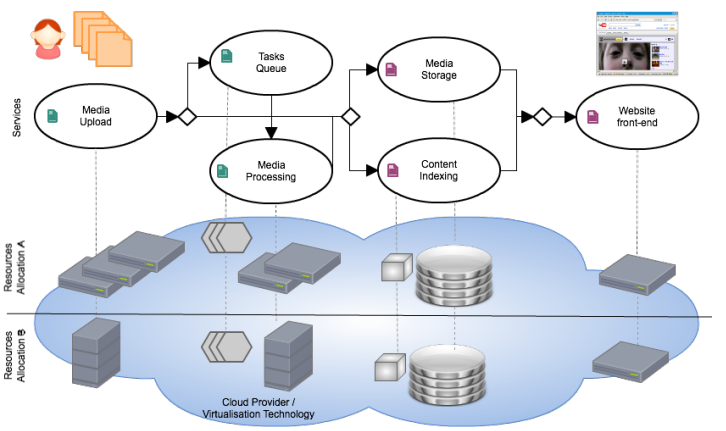
2. MOTIVATING EXAMPLE

Media sharing is one of the main Internet applications [Miller 2008]. This type of application was driven by the increasing use of social networks and content sharing platforms such as YouTube, Instagram and Facebook, and its growth brings scalability problems, with increasing demands for data storage and transfer, and the pressure to deliver faster service and other quality attributes. Cloud computing is therefore an increasingly used alternative for resource providers to circumvent these problems. Therefore, in this section we will explore some scenarios to illustrate the complexities involved in the management of scalability issues.

Let us suppose a fictitious organization that uses a public cloud provider or a datacenter (using some virtualization technology) to obtain resources for its applications. One of the applications consists in a choreography of services for media sharing on the web. This application comprises a service for media upload that communicates with other two services: one to perform media storage and another to perform media indexing. There is also a service that provides the website front end.



(a) Initial choreography.



(b) Adapted choreography.

Fig. 1. Choreography Deployment for the Media Sharing Application.

The services of the application have associated quality of service requirements: initially the upload service must handle at least 100 concurrent requests; data storage must be performed in a secure environment; and indexing overhead should be less than 1 second for at least 90% of the requests. Based on these requirements and on the expected demand, let us suppose that in order to enact the choreography, it is necessary to allocate one VM for the upload service, as well as using a scalable architecture for media storage, a relational database for data indexing, and another VM for the front end. The services and resources of the initial scenario are illustrated in Figure 1(a).

After choreography deployment and application execution starts, suppose there is a considerable increase in demand. This behavior is common in many web applications, which typically starts with only a handful of users but quickly grow to reach thousands and even millions of users. As an example, Facebook has an average growth of 250,000 new users per day [Facebook 2013].

In addition to increased demand, in our scenario another requirement was raised - the viewing of media in various formats. Accordingly, it is necessary to convert the original media, which is a intensive processing task. Thus, two new services must be added to the choreography: one to perform media processing before storage and another to control the queue for this. Furthermore, aiming to increase competitiveness, the upload QoS was modified, aiming to be able to handle ten times more requests concurrently. To meet this new scenario, it is necessary to review the initial resource allocation.

In the additional resource allocation we can adopt the strategies cited before: the first one is to do horizontal scaling. Accordingly, we can create other VM instances and get something like allocation A in Figure 1(b). On the other hand, we can use vertical scaling and keep the number of resources but increasing their configuration, as in the allocation B in Figure 1(b).

The main problem in this scenario is to decide which scalability strategy is the preferred option given the quality of service requirements and the cloud provider or technology features, e.g. performance, security, cost, etc. For instance, at a first glance horizontal scaling is a good choice for the media processing service due concurrency issues but what are the outcomes of adopting this strategy instead of vertical scaling? In addition we can even use both strategies by allocating more instances with an increased configuration. Another point is ability to determine the overhead as well as the ability to adopt each of these strategies when using a the given provider/technology.

3. SCALABILITY STRATEGIES IN CLOUD PLATFORMS

Current virtualization technologies allow the addition of new VM instances as well as the redimensioning of running VMs. For horizontal scaling, experiments have shown different values for VM startup time [Hill et al. 2010; Bellenger et al. 2011], although on average it takes about 1 minute, while vertical scaling allows to double the processing power in less than 1 second [Yazdanov and Fetzer 2012]. Gong et al. [Gong et al. 2010] indicate that changes in the amount of CPU take on average 120±0.55ms.

Another positive factor for the adoption of a hybrid approach rather than the commonly adopted practice of only scaling resources horizontally is that vertical scaling is quite advantageous in some cases, as Dawoud et al. [Dawoud et al. 2012] demonstrate. According to these authors, a web server running on a vertically scaled VM offers better performance than a web server running multiple VMs, i.e. a VM with 4 cores implies a lower response time compared to 4 VM instances running in parallel.

Table I presents details on automated resource scaling for some of the main providers and cloud technologies. As can be seen, the majority of them do not support automatic scaling but provide APIs that allow one to perform this task.

Table I. Scalability in cloud providers/technologies.

Cloud Provider / Technology	Automatic Horizontal Scaling	Automatic Vertical Scaling
Amazon (aws.amazon.com)	yes	no
Windows Azure (www.windowsazure.com)	yes	no
Google App Engine (developers.google.com/appengine)	yes	no
Google Compute Engine (cloud.google.com/products/compute-engine)	yes	no
Rackspace (www.rackspace.com)	no	yes
Flexiscale (flexiscaletechnologies.co.uk)	no	yes
GoGrid (www.gogrid.com)	no	no
Joyent Cloud (www.joyent.com)	no	yes
Eucalyptus (www.eucalyptus.com)	no	no
Xen (www.xenproject.org)	no	yes

By using the *Auto Scaling* and *Cloud Watch* in Amazon EC2, it is possible define policies for VM creation and destruction. The average startup time for a new instance on this provider is between 2

and 10 min, though this time is close to 100 sec for instances running Linux [Li et al. 2010]. Similarly, VM startup time in Windows Azure is around 10 min, although different requests for VM creation may take up different amounts of time. Experiments in [Hill et al. 2010] indicate a delay of 4 min between the startup time of the first and fourth instance using Azure. Some cloud providers like Google App and Compute Engine[1] offer horizontal auto scaling, although it is not possible manage its operation.

Regarding vertical scaling, in Rackspace, Flexiscale and Joyent Cloud it is possible to do scaling of processor but this requires VM reboot. CPU scaling can be performed until it reaches the full capacity of the underlying physical machine [Voorsluys et al. 2011]. In Xen we can also have scaling of memory, which is performed by a process known as *memory ballooning*, which allows changing of the amount of memory to a VM while it is running. A similar mechanism is offered for CPU scaling [Nathan 2013].

In addition to the APIs offered by cloud providers is possible to use frameworks to achieve automated scaling of resources. Table II [Ferry et al. 2013] shows some of these frameworks, together with the providers they support and details on the scaling capacity.

Table II. Frameworks for cloud resource management.

Framework	**Supported Providers**	**Scaling Strategy**
Cloudify (www.cloudifysource.org)	Amazon, OpenStack, Azure, HP Cloud, Rackspace	Automated scaling based on metrics related to VM configuration and number of instances.
Cloud Foundry (www.cloudfoundry.com)	Amazon, OpenStack, Rackspace, Eucalyptus	Manual change in the number of VMs associated with an application.
Scalr (www.scalr.com)	Amazon, OpenStack, Rack- space, Nimbula, Eucalyptus, IDC Frontier, CloudStack, Cloud Foundry	Automated scaling of infrastructure and database when they are overloaded or when scheduling is done.
OnApp (onapp.com)	Xen, KVM, VMware and other smaller public provider	Vertical scaling with automated VM migration when the physical machine does not have enough resources.

Besides scaling, it is necessary to take into account the time required to perform VM migration from one physical machine to another when the existing resources are insufficient. Considering Xen for instance, VM migration requires the transfer of all memory. However, the migration mechanism hides the latency by continuing running the application on the original VM while the memory contents are transferred. Experiments in [Ruth et al. 2011] show that the migration of a VM with 800MB of memory on a LAN using Xen caused unavailability of 165 ms to 210 ms and increased application execution time in 17-25 sec. However, during this period throughput decreases only 12%. The results in [Voorsluys et al. 2009] show that, in an instance of an almost overloaded system (serving 600 concurrent users), migration causes significant downtime (about 3 sec) but the 99^{th} SLA percentile could be met, i.e. 99% of more critical SLA could be met.

Based on the above, although there are no records of accurate results for all technologies in the literature, we can state that the achievement of scaling in two dimensions (vertical and horizontal) is feasible. Having said that, we now present some experiments to evaluate how these strategies work in the context of service choreographies.

4. EVALUATING SCALABILITY STRATEGIES TO SERVICE CHOREOGRAPHIES

We performed some preliminary experiments to analyze the scalability strategies and evaluate how it interferes in the resource usage and the QoS offered to choreographies. Through this experiment, we expect to answer the following research questions:

—Does the cost associated with the allocation of resources justify potential advantages obtained?

[1]In the Google Compute Engine horizontal auto scaling is implemented as an application from App Engine.

—Which one is the best strategy for resource scaling in a choreography enactment?

—How different scaling policies interfere in the usage of resources and non-functional characteristics of services choreographies?

The experiments were performed using simulations through a modified version of CloudSim [Calheiros et al. 2011]. Initially we present the modifications implemented and then we describe the experiments and their results. To design and describe the experiments we follow our structure proposed by Wohlin et al. [Wohlin et al. 2000].

4.1 Simulator

CloudSim [Calheiros et al. 2011] is an extensible toolkit that enables modeling and simulation of clouds, as well simulation of policies for resource provisioning. This simulator is usually used to investigate the infrastructure design decisions by analyzing different configurations [Caglar et al. 2013]. Cloud providers are modeled in the simulator as *Datacenters* receiving service requests. These requests are elements encapsulated in *VMs* that need to allocate shared processing power in a given *Host* in the datacenter. The *VMScheduler* component is responsible for scheduling the host that runs each VM.

Applications running in the cloud environment are represented as a set of *Cloudlets*, which store execution data as request size in millions of instructions (MI); and utilization modes of CPU, memory and bandwidth. The *DatacenterBroker* component is responsible for simulation and management of cloudlets, and to configure their policies of resource management.

CloudSim has some limitations that hinder the simulation of our scenario. For example, it does not enable the simulation of automatic resource provisioning and it simulates just a single service. Thus, we implemented an extension of CloudSim to overcome these limitations and enable the simulation of choreography enactment. The implementation of auto scaling mechanism was made using a simplified version of the model proposed by Amazon for its services *CloudWatch* and *Auto Scaling*. According to this model, it is possible to establish metrics for resource monitoring, and policies for VM manipulation.

For representation of choreographies, we adopted the semantics of Web Service Choreography Description Language (WS-CDL) [Kavantzas et al. 2005], an XML-based language to describe collaboration between multiple stakeholders in a process of business. WS-CDL is a W3C recommendation detailed at http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109.

The modifications made in CloudSim not influence the core operation, since we added only metrics collection and resource allocation policies. Thus, the results remain equivalent to a real scenario.

4.2 Simulation of Resource Allocation Policies Applied to a Service Choreography

We performed simulations of running the media sharing application in a cloud environment adopting different policies for resource allocation. In those simulations, we aimed to analyze how those policies impact in resource usage and the application QoS.

4.2.1 *Experiment Design.* Our experiment evaluates how different resource scaling policies interfere in the usage of cloud resources and the resulting choreography's QoS.

4.2.2 *Experiment Planning.* We evaluate the scalability by simulating horizontal, vertical and hybrid scaling policies. The simulated scenario is the enactment of the media sharing choreography (Section 2) in a single cloud provider. The following variables were analyzed:

—**Latency:** mean enactment time of the choreography.

—**Usage (VM):** mean and variance of the load in the virtual machines.

—**Usage (Host):** mean and variance of the load in the hosts (physical machines).

—**AWRT:** Average Weighted Response Time, as proposed by Grimme et al. [Grimme et al. 2008], which measures how much, on average, users should expect to have their requests met.

—**Execution Overhead:** how much, on average, the execution time differs from an estimated optimal value. This value is taken assuming that the request in question is the only one in the cloud, i.e., no other concurrent requests share the same resource.

The size of the media used as input in the simulation was obtained randomly, whereas values between 0 and 10 GB. The activities duration, expressed in MI, that composes the choreography was estimated according to the size of the media, considering the average connection speed in Brazil and values obtained with benchmarks [Movavi 2014; MySQL 2014; Seagate 2012].

We evaluated scenarios with different requests per second rates: 1, 5, 10 and 30. For the horizontal scaling we performed scenarios with 1, 5, 10 and 30 running VMs. We used VM configurations equivalent to the types of Amazon EC2 *m1.small*, *m1.medium*, *m1.large* and *m1.xlarge* for the vertical scaling. In the hybrid approach, both the amount and configuration of VMs have changed. For each simulation, we consider a fixed amount of resources allocated. The simulated cloud consists of 10 hosts with same configuration: equivalent to a machine Intel® Xeon® Processor X5570, 8 GB of RAM.

4.2.3 *Experiment Execution.* This experiment is based on the modified version CloudSim described above. The choreographies submitted in the simulation were generated and stored so that the same input sets were used in the three scaling strategies.

There was no need of any treatment regarding the validity of the data.

4.2.4 *Analysis and Interpretation of Results.* Each experiment starts with the same scenario of one VM type *small* and load of one request/s. To evidentiate the difference among results, charts were plotted on a logarithmic scale, except for the chart of strategy costs (Figure 3).

The first variable taken was the latency. As can be seen in Figure 2(a), when there is 1 req/s the increase in VM number only brings high gains when going from 1 to 5 VMs, with a decrease of 52.8% on average latency. With scaling to 10 VMs the gain decreases to 4.14% and after that no more gains are obtained. This is due the fact that there are few requests, since a similar behavior is observed in horizontal scaling when there are 5 req/s. Therefore, the horizontal scaling can only be justified for large-scale scenes, such as 30 req/s, where the gain is always greater than 50% (Figure 2(a)).

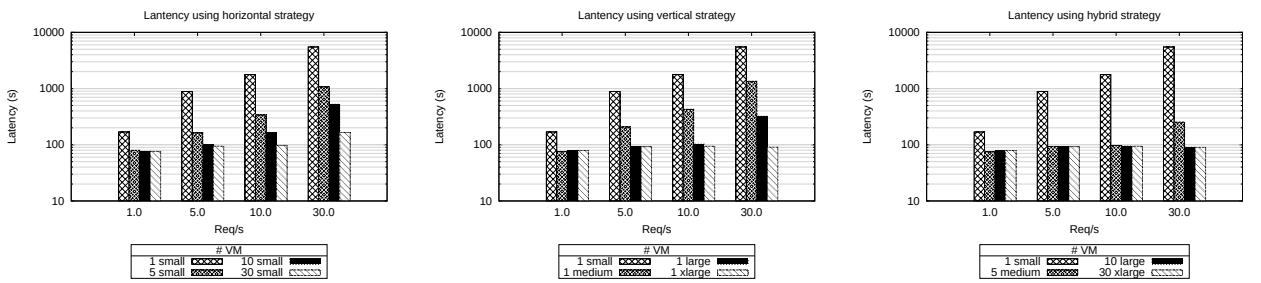


Fig. 2. Latency. (a) Horizontal strategy. (b) Vertical strategy. (c) Hybrid strategy.

Comparing the horizontal and vertical approaches (Figures 2(a) and 2(b)), we can see that, regarding latency, to use only 1 VM type *medium* is almost as satisfying as using 5 VMs type *small*. For types *large* and *xlarge* the vertical strategy is more advantageous than the horizontal strategy. This shows

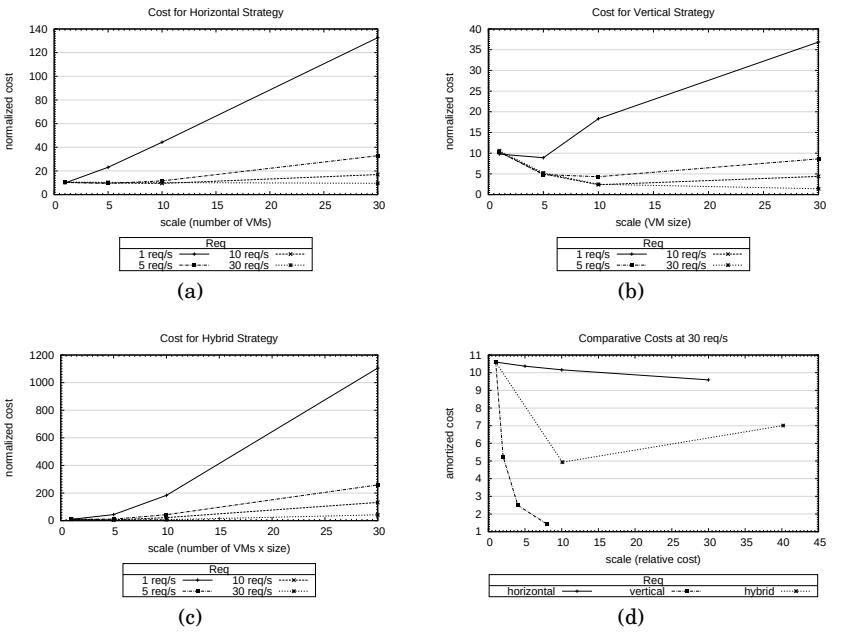


Fig. 3. Cost of scaling. (a) Horizontal strategy. (b) Vertical strategy. (c) Hybrid strategy. (d) Comparative among strategies.

that the allocation of a larger number of VMs is not the best approach in this scenario. This is mainly because of the cost: considering a public cloud, the most extreme case (30 req/s) would require the expenditure of $2.40 per hour in a horizontal approach and $0.64 using the vertical strategy[2].

The hybrid strategy (Figure 2(c)) shows that increasing the amount and configuration of VMs only bring benefit in the first modification, i.e. move from one VM type *small* to 5 VMs type *medium*. The only exception is the case of 30 req/s, for which the second modification also brought gains. Comparing the vertical and hybrid approaches (Figures 2(a) and 2(b)) we found that the hybrid strategy has not brought gains in some cases. This reinforces the argument that changing the configuration of VMs can be a better strategy than allocate a greater amount of VMs.

To compare each scaling strategy, we defined the cost of each scaling, called *normalized cost*, by the expression $cost\_per\_hour/(QoS \times load)$, where $cost\_per\_hour$ is the cost to implement the strategy, considering the number and type of VMs, QoS is the inverse of latency and $load$ is the number of requests per seconds. Thus, a scaling strategy is better when the normalized cost decreases as the scale increases, for a same load. To a fair comparison among different strategies, we converted each strategy scale to a relative scale cost (por hour), considering that one small VM has a cost of one unit.

Figure 3 shows graphs of the normalized cost for each scaling strategy. Graph 3(d) shows a comparison between the cost of each strategy, on a load of 30 request/s. Graph 3(d) endorses the idea that thet vertical strategy is the best strategy for the simulated scenario. Also, each strategy presents a point where it does not produce any benefit. As shown in graphs 3(a) and 3(c), scaling in horizontal and hybrid approaches is only a beneficial approach at higher loads (close to 30 request/s).

The execution overhead and AWRT presented the same behavior of latency, so the corresponding charts will not be shown or discussed. We also analyze the impact of scaling approaches on the use of

[2]This estimation is using Amazon EC2 resources in São Paulo (Brazil) availability zone in February/2014.

resources, since the use of resources must be maximized in private clouds. Initially we analyze the use of physical machines. Since this factor depends solely on the amount and configuration of VMs created and there is no change in these attributes at runtime, the results are similar for all the requests rate and, so we put them together to facilitate comparison.

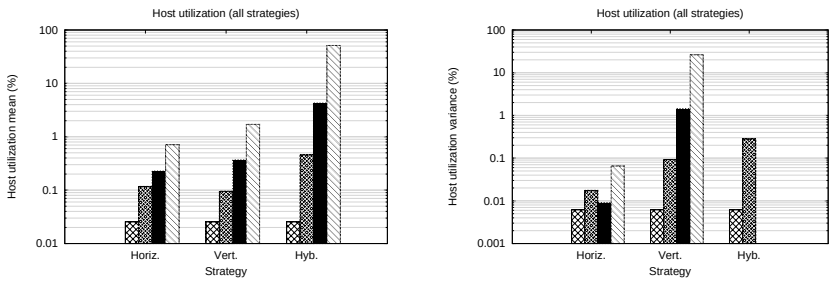


Fig. 4. Host usage. (a) Average. (b) Variance.

In Figure 4 we can see the average and variance of utilization of physical machines. As showed in Figure 4(a), the average host usage has no major changes when comparing the horizontal and vertical approaches, despite a much larger amount of VM created in the horizontal approach. Regarding the hybrid approach, only there was considerable use of resources in the latter case (30 VMs type *xlarge*), where the average utilization of physical machines was above 50%. The low utilization rates for the other cases is due to the large number of available resources in the cloud.

The usage variance (Figure 4(b)) shows that resources were allocated almost uniformly. The only exception occurred in the vertical approach using VM type *xlarge* for which variance was 26.31%.

Regarding the VM usage, we note that even with a greater number of instances, the horizontal approach has a higher average utilization (Figure 5(a)). This was expected since, by having a more limited setting, VMs need to conduct a more intensive processing to complete the activities. On the contrary, the creation of 30 VMs makes the average utilization of VMs lesser than 2% for 1, 5 and 10 req/s. Even at 30 req/s the usage is no more than 25%, which shows that most of these resources are idle, representing a loss. On the other hand, use only 1 or 5 VMs (with same configuration) causes the average utilization be near 100% when there are many requests.

The vertical approach also seemed to be the best alternative with respect to the use of resources. Mean usage (Figure 5(b)) with the type *medium* was around 50% in almost all cases. For the type *large* this value was lower, not exceeding 25%. The use of type *xlarge* makes the average utilization be very low, less than 4% in all cases. For the hybrid case (Figure 5(c)) the scaling only improves substantially the usage from the first to the second simulated case (5 VMs type *medium*), since allocation of more resources decrease the average usage to a low percentage, no more than 1%.

In the analysis of usage variance it makes no sense to take into account the cases with only one VM running. As a result, according to Figures 5(d) and 5(e) we can see that there are major differences only in extreme cases where the amount of VMs is greater than or equal to 10. This happens because in the other cases there is a limited number of VM, which makes usage occurs almost uniformly.

With this experiment we concluded that the allocation of a greater number of VMs is not the best strategy when considering services choreographies. Our results suggest that, due to dependencies between choreography activities, the use of a strategy that minimizes the execution time may produce improved results. However it is still necessary a mechanism to decide which strategy is the best in each

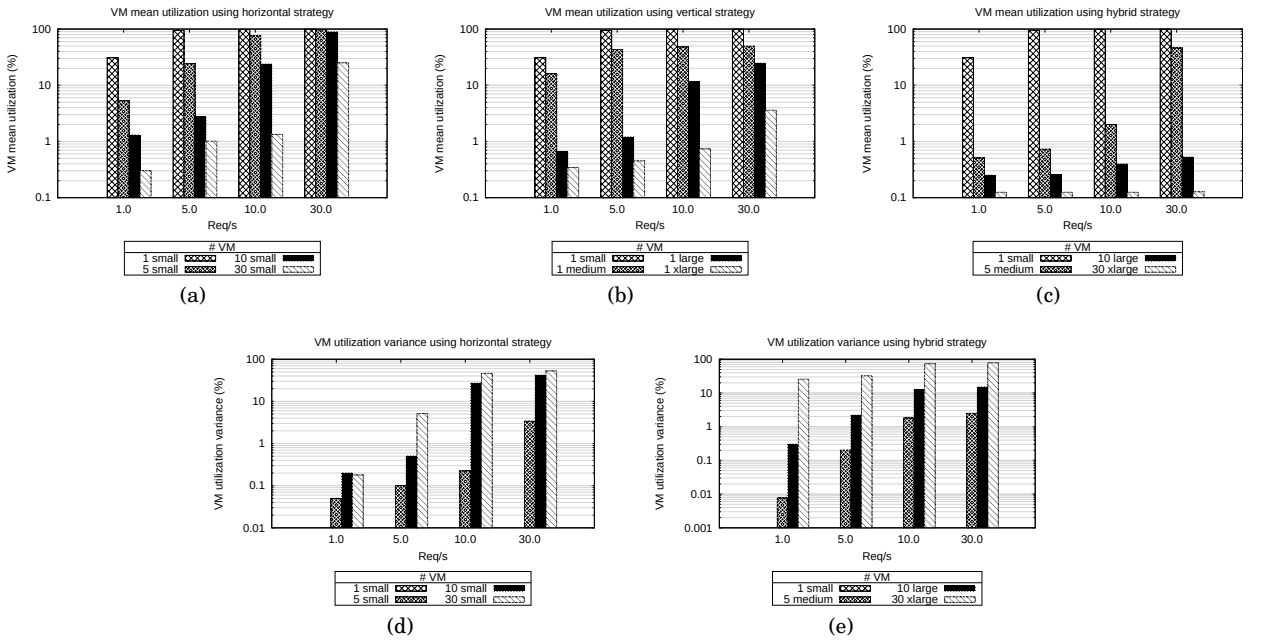


Fig. 5. VM usage. Average on horizontal (a), vertical (b) and hybrid (c) strategy. Variance on horizontal (d) and hybrid (e) strategy.

scenario, using adaptive algorithms that learn with the execution historic. For services choreographies we can even use different strategies to each services set. In a future experiment, we plan to evaluate how a choreography style and architecture may influence the gain of each scaling strategy.

5. RELATED WORK

There have been significant research efforts on the analysis of virtualization technologies [Huang et al. 2013; Voorsluys et al. 2009] or cloud providers performance [Hill et al. 2010; Dejun et al. 2010], mainly related to effects of horizontal or vertical scaling on the perceived latency.

Vaquero et al. [Vaquero et al. 2011] survey various approaches for application scalability in clouds at three different levels: server, network and platform; they also discriminate the two types of scalability: horizontal and vertical.

Suleiman et al. [Suleiman et al. 2012] identify a series of research challenges related to the scalability of existing solutions. Their work aims to determine how much to scale, taking into account automated scaling mechanisms and the costs associated with licensing, as well as the flexibility enabled by the size and type of resources that can be scaled. They also analyzed how to scale, and which scaling strategy to choose, conducting a trade-off analysis between horizontal and vertical solutions.

Nevertheless none of these works considers scalability in clouds to enact service choreographies. As we pointed out above, there are some particularities that must be taken into account, like dependencies between services or concurrency issues. To the best of our knowledge there is no other work that considers this subject.

6. FINAL REMARKS

In this paper we discuss how the main cloud providers and virtualization technologies provide scalability. By means of experiments carried out through simulation, we investigate the impacts of using three scalability strategies do enact service choreographies on cloud resources.

While vertical scalability is possible in principle for all applications, it largely depends on the service provider to offer the mechanisms to implement this type of scaling dynamically. Horizontal scalability on the other hand mostly depends on the application components and the application as a whole to support it as an option [Andrikopoulos et al. 2013]. As pointed out the overhead of each strategy also needs to be taken into account. Horizontal scaling, for instance, requires on average 10 min to start a new VM instance, which may not be feasible in scenarios that involve real-time applications.

We evaluated some scalability scenarios to enact service choreographies. In our analysis we concluded that vertical scaling is the best option in terms of cost, resource usage and application QoS attributes for the application considered. This result, although not be general for various application areas, it has applicability in resource allocation in approaches that use cloud computing in the general case. For instance, a model-driven development process can manage the relation between the infrastructure and the actual application non-functional requirements choosing the most suitable scalability strategy to meet these requirements [Gomes et al. 2013].

Nevertheless important scenarios with scalability patterns still need to be evaluated. In addition, a more precise analysis must be performed in order to provide elements that will enable a more effective scalability strategy. In particular, it is necessary to characterize the nature and behavior of choreographies, including the characteristics of each individual service, using this information to refine the evaluation of scalability strategies. In our work the main focus was application load but this analysis can take into account other aspects such as service load and the use of public vs. private clouds.

REFERENCES

Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, and Steve Strauch. 2013. How to adapt applications for the Cloud environment. *Computing* 95, 6 (2013), 493–535. DOI: http://dx.doi.org/10.1007/s00607-012-0248-2

Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The internet of things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.

Adam Barker, Christopher D Walton, and David Robertson. 2009. Choreographing web services. *Services Computing, IEEE Transactions on* 2, 2 (2009), 152–166.

Dominique Bellenger, Jens Bertram, Andy Budina, Arne Koschel, Benjamin Pfänder, Carsten Serowy, Irina Astrova, Stella Gatziu Grivas, and Marc Schaaf. 2011. Scaling in cloud environments. *Recent Researches in Computer Science* (2011).

Gordon Blair, Fabio Kon, Walfredo Cirne, Dejan Milojicic, Raghu Ramakrishnan, Dan Reed, and Dilma Silva. 2011. Perspectives on cloud computing: interviews with five leading scientists from the cloud community. *Journal of Internet Services and Applications* 2, 1 (2011), 3–9. http://dx.doi.org/10.1007/s13174-011-0023-1

Faruk Caglar, Kyoungho An, Shashank Shekhar, and Aniruddha Gokhale. 2013. Model-driven performance estimation, deployment, and resource management for cloud-hosted services. In *Proceedings of the 2013 ACM workshop on Domain-specific modeling*. ACM, 21–26.

Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience* 41, 1 (2011), 23–50.

Lawrence Chung and Julio Cesar Sampaio do Prado Leite. 2009. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*. Springer, 363–379.

P Daras, D Williams, C Guerrero, I Kegel, I Laso, J Bouwen, J Meunier, N Niebert, and T Zahariadis. 2009. Why do we need a content-centric future Internet? proposals towards content-centric Internet architectures. *Information Society and Media Journal* (2009).

Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel. 2012. Elastic virtual machine for fine-grained cloud resource provisioning. In *Global Trends in Computing and Communication Systems*. Springer, 11–25.

Jiang Dejun, Guillaume Pierre, and Chi-Hung Chi. 2010. EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, Asit Dan, FrŐdŐric Gittler, and Farouk Toumani (Eds.). Lecture Notes in Computer Science, Vol. 6275. Springer Berlin Heidelberg, 197–207. DOI: http://dx.doi.org/10.1007/978-3-642-16132-2_19

Facebook. 2013. Statistics. *https://newsroom.fb.com/* (2013).

Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. 2013. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *CLOUD 2013: IEEE 6th International Conference on Cloud Computing*. 887–894.

Raphael Gomes, Fabio Costa, and Nelly Bencomo. 2013. On Modeling and Satisfaction of Non-Functional Requirements using Cloud Computing. In *Proceedings of the 2013 IEEE Latin America Conference on Cloud Computing and Communications (LatinCloud 2013)*.

Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*. IEEE, 9–16.

Christian Grimme, Joachim Lepping, and Alexander Papaspyrou. 2008. Prospects of collaboration between compute providers by means of job interchange. In *Job Scheduling Strategies for Parallel Processing*. Springer, 132–151.

Zach Hill, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez, and Marty Humphrey. 2010. Early observations on the performance of Windows Azure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 367–376.

Xiaofei Huang, Xiaoying Bai, and R.M. Lee. 2013. An Empirical Study of VMM Overhead, Configuration Performance and Scalability. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*. 359–366. DOI: http://dx.doi.org/10.1109/SOSE.2013.72

Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. 2005. Web services choreography description language version 1.0. *W3C candidate recommendation* 9 (2005).

Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 1–14.

Michael Miller. 2008. *Cloud computing: Web-based applications that change the way you work and collaborate online*. Que publishing.

Movavi. 2014. Faster Performance with Cutting-Edge Tech. *http://www.movavi.com/videoconverter/performance.html* (2014).

MySQL. 2014. Estimating Query Performance. *http://dev.mysql.com/doc/refman/5.0/en/estimating-performance.html* (2014).

Senthil Nathan. 2013. *Dynamic Resource Provisioning for Virtual Machine through Vertical Scaling and Horizontal Scaling*. Ph.D. Dissertation. Department of Computer Science and Engineering, Indian Institute of Technology.

Dimitri Papadimitriou and others. 2009. Future Internet–the cross-ETP vision document. *European Technology Platform, Alcatel Lucent* 8 (2009).

Paul Ruth, Junghwan Rhee, Dongyan Xu, Sebastien Goasguen, and Rick Kennell. 2011. Autonomic live adaptation of virtual networked environments in a multidomain infrastructure. *Journal of Internet Services and Applications* 2, 2 (2011), 141–154.

Seagate. 2012. *Savvio® 10K.5 SAS Product Manual*.

Basem Suleiman, Sherif Sakr, Ross Jeffery, and Anna Liu. 2012. On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. *Journal of Internet Services and Applications* 3, 2 (2012), 173–193. DOI: http://dx.doi.org/10.1007/s13174-011-0050-y

Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. 2011. Dynamically Scaling Applications in the Cloud. *SIGCOMM Comput. Commun. Rev.* 41, 1 (Jan. 2011), 45–52. DOI: http://dx.doi.org/10.1145/1925861.1925869

Hugues Vincent, Valérie Issarny, Nikolaos Georgantas, Emilio Francesquini, Alfredo Goldman, and Fabio Kon. 2010. CHOReOS: scaling choreographies for the internet of the future. In *Middleware'10 Posters and Demos Track*. ACM, 8.

W. Voorsluys, J. Broberg, and R. Buyya. 2011. Introduction to cloud computing. *Cloud Computing* (2011), 1–41.

William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. 2009. Cost of virtual machine live migration in clouds: A performance evaluation. In *Cloud Computing*. Springer, 254–265.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2000. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

Lenar Yazdanov and Christof Fetzer. 2012. Vertical scaling for prioritized VMs provisioning. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*. IEEE, 118–125.

Received May 2014; revised Jun 2014; accepted Jul 2014