Cloud Elasticity: going beyond demand as user load

Cristian Chilipirea, Alexandru Constantin, Dan Popa, Octavian Crintea, Ciprian Dobre
Faculty of Automatic Control and Computers
University Politehnica of Bucharest
Bucharest, Romania
Email: {cristian.chilipirea, ciprian.dobre}@cs.pub.ro

***Abstract*—Cloud computing systems have become not only popular, but extensively used. They are supported and exploited by both industry and academia. Cloud providers have diversified and so did the software offered by their systems. Infrastructure as a Service (IaaS) clouds are now available from single virtual machine use cases, such as a personal server, to specialized high performance or machine learning engines. This popularity has been brought by the low-cost and risk-free feature of renting computing resources instead of buying them, in a large, one-time investment. Furthermore, clouds permit their clients the use of elasticity.**

Elasticity is the most relevant feature of cloud computing. It refers to the clients' ability to easily change the number of rented resources in a live environment. This permits the entire system to handle differences in load. Most cloud clients serve web applications or services to third parties. In these cases, load differences can be correlated to the number of users for the service and elasticity is used to handle differences in what is called "user load". Most of the scientific literature approaches elasticity looking solely at "user load". To give a clearer understanding, the majority of cloud frameworks in use today work as follows: they start a number of worker nodes, and tasks are assigned to them for execution. Only when the "user load" changes, the number of workers is adjusted, if any.

In this paper, we propose an alternative approach, where the number of workers depends on the actual requirements coming from the different execution steps of an application. We show such an idea can be achieved for several workflows from different fields and that it can bring significant benefits to execution time and cost.

Keywords: elasticity, cloud computing, scheduling, cost

I. Introduction

Cloud computing [1], [2] has changed the ITC industry. Companies like Amazon[1], Google[2] and Microsoft[3] have built large computing clusters of as many as 1 million servers[4]. Each server can run multiple virtual machines and these virtual machines are rented to customers. This type of service, where virtual machines are rented, is called Infrastructure as a Service (IaaS). This paper focuses on IaaS clouds, since the mechanism we propose best fits the needs of IaaS clients. However, it can easily be applied to Platform as a Service (PaaS) and Software as a Service (SaaS).

Cloud computing did not innovate with the idea of renting compute resources. This was previously done by multiple companies, usually telecom companies which had small clusters and large bandwidth access. The innovative factor, and still the most important in cloud computing is elasticity. **In cloud computing elasticity represents the ability to vary the number of resources** (worker nodes in the case of IaaS) **depending on demand**. When NIST defined cloud computing [3], it listed "Rapid elasticity" and "On-demand self-service" (the enabler of elasticity) as essential characteristics.

In the context of cloud computing "demand" is not clearly defined and usually holds the meaning of "user load". An obvious example would be Netflix[5], a cloud client for Amazon[6], which offers streaming services to third parties. The third parties, clients for Netflix, represent the users in "user load", number of concurrent clients accessing the service. Netflix (and most cloud clients) uses **elasticity to enable applications which continuously receive inputs** (user requests for Netflix) **and generate outputs** (video streams for Netflix) **to vary the number of resources depending on the number of inputs over time, "user load"**. In the related work, Section II, we show that in most scientific papers that tackle elasticity "demand" takes this form.

We propose an alternative use case for elasticity and cloud computing. **Take applications whose execution can be separated in multiple steps and different steps require different optimal numbers of resources. Using elasticity these applications can execute having the optimal number of resources for each individual step**. Here, "demand" is given by the application and not by outside factors like "user load".

There are many applications whose execution can be split into multiple steps (we offer a few examples in Section III). A large class of this type of applications is the one where execution can be split into Directed Acyclic Graphs (DAG). DAG workflows have received a lot of attention in the scientific literature, especially with regards to scheduling [4]. These are applications that so far do not make use of cloud elasticity. Using our method this can change.

II. Related Work

All definitions of elasticity state that it is dependent on demand, without going in detail of what demand is: When NIST defined cloud computing [3] they explained the essential

[1]https://aws.amazon.com/ec2/
[2]https://cloud.google.com/compute/
[3]https://azure.microsoft.com/en-us/
[4]https://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx
[5]https://www.netflix.com/
[6]https://aws.amazon.com/solutions/case-studies/netflix/

characteristic "rapid elasticity" as *"Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with **demand**"*; In [6] the authors define elasticity as *"the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current **demand** as closely as possible"*; Another definition [5] describes the need of cloud systems to be "highly scalable" as *"Infrastructure providers pool large amount of resources from data centers and make them easily accessible. A service provider can easily expand its service to large scales in order to handle rapid increase in service **demands** (e.g., flash-crowd effect)."*

The purpose of elasticity as defined in [7] is to provide: performance, cost, increase infrastructure capacity, energy. All of these can be related to "user load".

In the literature we found elasticity being used to take advantage of:

a) Available resources: when not all resources are available from the start, elasticity can be used to permit an application to take advantage of resources that become available later in its execution. In [9] the authors introduced elastic algorithms. Elastic algorithms are a class of algorithms which depending on the amount of resources at their disposal provide a different level of accuracy for the results. Algorithms that provide results with different accuracy depending on execution time have been addressed before, but the idea that the accuracy can change with the amount of resources is new. In contrast [10] take resource usage maximization as the main factor when dealing with multi-tenant clusters. They propose that cluster utilization should be maximized and that this can be achieved by elastically varying the number of resources for each application, even when the number of applications changes in time. The authors of [11] show that the number of used resources can be changed during execution even when the application is not designed to take this into account. They start a distributed application on multiple virtual machines, all on the same physical machine, and when more physical resources are available or the demand increases, they migrate some of the virtual machines to newly booted physical machines.

b) Heterogeneous systems: When machines offered by the cloud provider are not identical application can be built to take advantage of the different resources. This need can be both in the form of hardware (machines with specialized processors such as GPUs) and software (different libraries or frameworks). In their article [12] propose the use of a set of rules with which to make use of elasticity to schedule different parts of the application to different machines in the cloud.

c) Cost: Kingfisher [13] is a cloud scheduling system for the user which is cost-aware. It makes use of elasticity to increase and decrease the number of machines according to both demand and cost.

The rest of the literature which tackles elasticity only considers "user load": Sybl [14] proposes a language with which to better control elasticity; other works try to offer ways of measuring elasticity [8] similarly to how physical elasticity works or [20] which offer a penalty measure for when resource need is not perfectly matched with available resources; There are benchmarks for elasticity [15]; Cloudscale [16] tries to predict the load of a system to better enable elasticity; [17] provides a mechanism to model check it; even in works which addresses hybrid cloud scenarios [18] and multi-tier cloud applications [19].

The realation between elasticity and "user load" holds true even in surveys on elasticity [21] and in articles discussing programming for elasticity [22]. With the few exceptions, the scientific literature regarding cloud elasticity mainly concentrates on varying the number of resources with "user load". As to our knowledge we are the first to propose the idea of varying the number of resource depending strictly on the requirements of different steps of one application.

III. Case studies

In order to show how elasticity can be used in scenarios other than the ones dependent on "user load" we present a number of proof of concept applications that require a different amount of resources (machines) in subsequent steps of their execution. These applications are chosen from different fields in order to emphasize that this method can be applied in a large number of cases. In the next subsections we go through each of the applications and we show how we split their respective execution flow in distinct steps. A step is a distinct part of the application which takes a set of inputs (possibly from a different steps) and creates a set of outputs (which can be used as inputs for other steps).

For every application, each step is run on multiple machines. In the paper we do not present the execution times but the speedup obtained for different number of machines. The speedup is defined "for each number of processors k as the ratio of the elapsed time when executing a program on a single processor (the single processor execution time) to the execution time when k processors are available", according to [23]. In other words, $S(k) = T(1)/T(k)$, where S is speedup and $T(k)$ is the execution time of the process on k machines. Execution time on one machine can vary with several orders of magnitude from one step of the same application to the other. By using speedup, we can present the results for all steps of an application in the same figure. Execution time is actually irrelevant to this article as it is dependent on the application, it's implementation, optimizations and the hardware. The goal here is not to achieve a minimal total execution time. This problem has been addressed for each of the applications we present. The goal is to determine the optimal number of machines needed for each step of different application. For this, speedup is sufficient.

In order to test the applications, we used different environments ranging from our compute cluster (from which we used up to 12 Nehalem machines with Intel Xeon X5657 CPUs) to cloud virtual machines rented from Amazon and Microsoft. In all scenarios we used distributed frameworks,

such as MPI [7] and the applications were run using multiple machines, physical in the case of our cluster and virtual in the case of actual cloud systems.

A. Web crawler

A web crawler application represents the core of web search systems such as Google[8] or Bing[9]. It is a type of internet bot that browses web pages on the internet in order to extract information. In the case of search engines this information is used to create a searchable index. Because of the sheer number of web pages, in the tens of billions (according to World Wide Web Size[10]), a web crawler needs to use multiple machines to process even a small part of the accessible web pages on the internet.

We built a simple web crawler, similar to the one proposed in [24], but without some features like page ranking or lexicon. The goal of our web crawler is to generate a list of the most common words on the web. To achieve this, it needs to download web pages, remove duplicate URLs and calculate word frequencies. Each of these steps can be distributed to execute across multiple machines. The application was built using Scrapy framework[11].

d) Step 1 - Web page download and link extraction: A repository consisting of a list of web links (also known as Uniform Resource Locator - URL) needs to be shared across all worker machines. Each machine selects a URL from the central repository, downloads the web page that corresponds to the URL, parses the web page and extracts all the links as well as all the words in the web page. The extracted links are added to the central repository, from where they wait to be selected. The words are added to a separate database. Figure 1 shows the architecture. In theory the execution ends when the entire "crawlable web" is processed. In practice there are many limitations, such as the storage space for the shared repository. We stopped our experiments after 100,000 web pages have been processed.

A web crawler is dependent on the initial list of URLs submitted to the shared repository. We used as an initial seed the DMOZ website[12]. The website is a human-edited directory of the Web.

e) Step 2 - Duplicate URL filtering: The URL list from the central repository is copied to each worker machine. With N worker machines each worker is responsible of $1/N$ of the entire list. The worker compares each of the elements it is responsible for with all the elements in the list. When duplicates are detected one of them is deleted. The final list is merged and copied back to the central repository. There are of course more efficient solutions for the problem.

[7]https://www.mpi-forum.org/
[8]http://www.google.com
[9]http://www.bing.com
[10]http://www.worldwidewebsize.com/
[11]http://scrapy.org
[12]http://www.dmoz.org

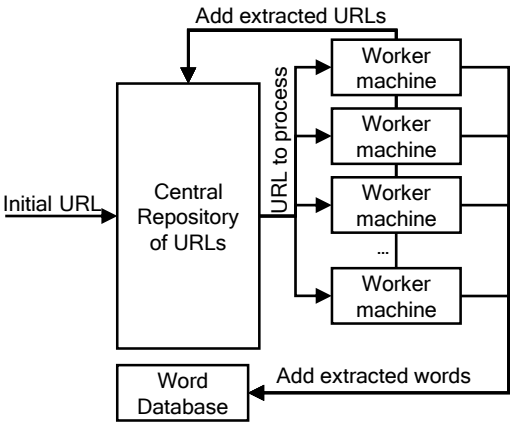

Fig. 1: Architecture of a web crawling engine

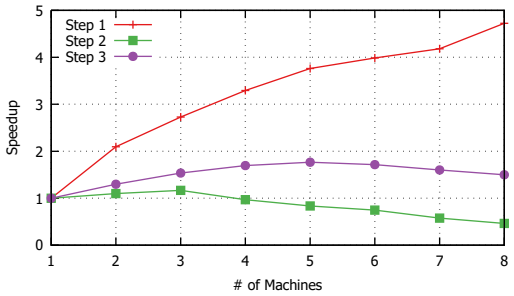

Fig. 2: Speedup - Web Crawler

f) Step 3 - Word frequency calculator: Similar to the duplicate URL filtering the list with all of the words is copied to the worker nodes. Each worker is responsible for $1/N$ of the words in the data set. When duplicates are identified the counter for that word is increased. The data from all machines is merged in a list of words and their frequencies.

After we built the application we performed our experiments. We measured the average execution time per item, for the 100,000 web pages. We used the average execution time per item to calculate the speedup for different number of machines. We performed the experiments, for each step, with 1 to 8 machines. The results are presented in Figure 2 and we can easily conclude that the speedup values differ for each step of the application. In Section IV we go into detail on how these differences permit the use of elasticity.

B. Machine learning - Q-learning

Q-learning [25] is a reinforcement learning technique. Reinforcement learning represents a part of machine learning focused on how agents need to take actions in order to maximize a reward. Q-learning is used to determine an optimal action-selection policy for Markov Decision Processes [26].

A Markov Decision Process (MDP) is composed of a set of states S, a set of actions A, a set of probabilities $P_a(s,s') = Pr(s'|s,a)$ that applying action a in state s causes a transition to state s', a set of rewards $R(s)$, and a discount factor $\gamma \in [0,1]$

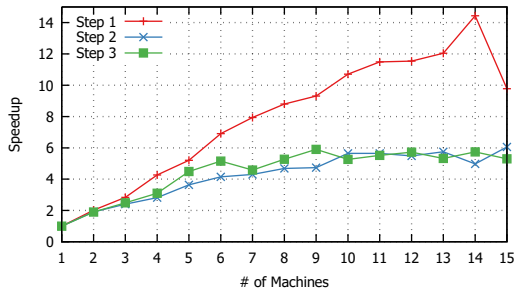

Fig. 3: Speedup - Q-learning

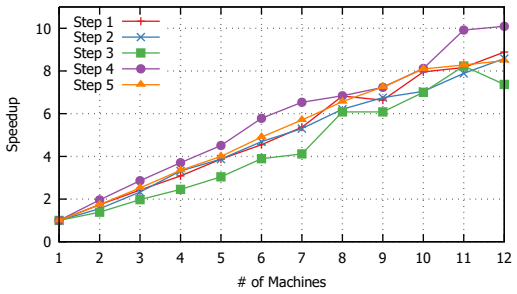

Fig. 4: Speedup - Render Pass Noise Reduction

which determines the importance of long term rewards in the detriment of short term ones.

The scope of Q-learning is to calculate the quantity Q of all state-action combinations. This is achieved by applying equation 1. The initial values Q_0 are usually set to be random.

$$Q_{i+1}(s,a) = (1-\alpha)Q_i(s,a) + \alpha(R(s) + \gamma \max_a Q_i(s',a)) \quad (1)$$

g) Step 1 - Generate Q_0: Because Q can be initialized with random values we chose to initialize it using a greedy approach. The algorithm goes through all the states and records which transitions have the highest probability. These are used to generate Q_0.

h) Step 2 - Calculate Q: Using equation 1 we calculate Q. Because the work is split across multiple machines different processes are responsible for different states. This means that there are conflicts where the information of state s' is not available. All conflicts are noted.

i) Step 3 - Solve conflicts: The list with all the conflicts is split across machines. Each machine solves a set of conflicts and returns the result to the controller which merges them in the Q matrix. The conflicts are solved by applying equation 1. Step 2 and 3 need to be applied multiple times in order for the values of the Q matrix to converge.

We tested the algorithm on a MDP that models a temperature control system. The MDP has 2001 states that represent temperature (from $0°C$ to $20°C$) in increments of a hundredth of a degree and 2001 actions which represent the power of the actuator (from -100% - cooling to 100% - heating) in increments of a tenth of a percent. The training processed involved 100 executions of steps 2 and 3.

The average execution time for each step is used to calculate the speedup. Figure 3 presents the speedup of each of the steps given between 1 and 15 machines. In Section IV we go into detail on how these differences permit the use of elasticity.

C. Rendering

During a rendering process multiple passes are executed that create a final image. Each pass has a specific role such as the creation of glossy surfaces or depth detail. A method to reduce noise is to take the final image and blur it. In some cases, edge detection is involved, but even then edges are blurred.

Render pass noise reduction[13] (RPNR) is an algorithm that reduces noise only within given surfaces. It uses different passes of the rendering process in order to obtain the desired result. It requires the Combined image (final image), the Ambient Occlusion, the Diffuse Color pass, Normal and Z (depth) passes. All steps take as input one or more images and output one image. They can all be easily distributed by splitting the input images and merging the output ones.

j) Step 1 - Blurring combined image: The result will be used as a *delimiter* along with the other passes (except the Ambient Occlusion one). It is meant to preserve Glossy detail when a Glossy pass is not available.

k) Step 2 - Transform Ambient Occlusion: The Ambient Occlusion pass is blurred, transformed to greyscale and inverted. It is used to determine the areas with the most noise.

l) Step 3 - Z (depth) to grayscale: The Z pass is converted to grayscale. It is used along with the Normal pass to attenuate edge noise.

m) Step 4 - Blurring surfaces: Here blurring is done differently than a standard blur. Instead of weighting surrounding pixels by distance they are weighted by how much they differ from the original image based on the *delimiters*. Pixels that are too different are ignored.

n) Step 5 - Blurring edges: During the previous step the edges are left untouched and contain the original noise. Another blur is executed similarly to step 4. In this case the difference between the original and the output image of step 4 is used. For the pixels where the difference is 0 and are detected to be edges by the Normal and Z pass, a 50-50% blend between their values and the combined image ones.

Steps 1, 2 and 3 are independent on each other and can be executed in parallel. However, their output is used by step 4 and step's 4 output is used in step 5. The execution time was measured for each step independently. Using the execution time we calculated the speedup, the results are available in Figure 4. These results are extremely dependent on the data set. Operations such as converting an image to grayscale are embarrassingly parallel. The optimal number of machines increases with the size of the input image. This is why graphic processors have so many cores. In Section IV we go into detail on how these differences permit the use of elasticity.

[13]https://github.com/lmmenge/RPNR

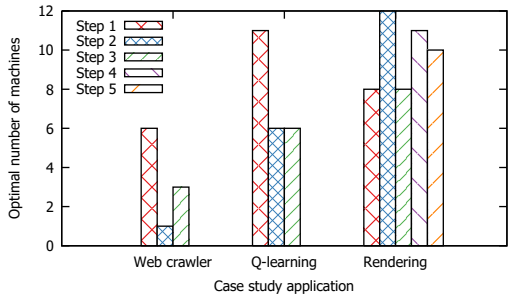

Fig. 5: Optimal number of machines for the case study applications

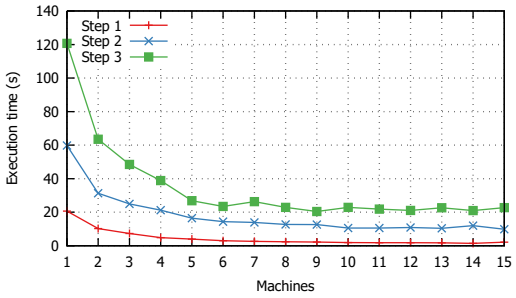

Fig. 6: Execution times for the Q-learning application

IV. Experimental Results

There are multiple policies that can be used to determine the optimal number of machines (resources) on which to run a distributed process [27]. These policies are usually dependent on cost, energy consumption or dead-lines. We propose a simple example policy that uses equation 2, which can uniformly be used on all the applications we presented and it does not depend on outside factors such as cost. Here S represents speedup and k the number of resources. Using the equation, we can set the optimal number of resources to be equal with the biggest number of resources as long as adding another resource offers a large enough (bigger than ε) speedup difference. We chose ε empirically to be 0.2. Finding the optimal number of machines in this manner is far from optimal, it serves as an example of a policy that can be used with our method. A better one would consider execution time, because speedup hides large variations in execution time that can be exploited for further gains.

$$optimal\_\#\_of\_machines = max(\{k|S(k+1) - S(k) < \varepsilon\}) \quad (2)$$

By using equation 2 on the speedup values from the case studies in Section III we calculated the optimal number of machines on which to execute each step of the applications. We present the results in Figure 5 in which we can observe that for all applications there are different steps that require different numbers of machines. Using cloud elasticity and these results one can run the applications in a way in which they would always execute on the optimal number of machines, regardless of which is the current step in the execution. This method can be easily used for many other applications. There are many examples of applications whose execution already take the form of a workflow.

In order to show the benefits of our method we take a closer look at the Q-learning application. The execution times for this application are presented in Figure 6. We use execution times from the figure and calculate cost considering the application is run on one machine, on the maximal optimal number of machines and using our method. As a simplification, we consider the cost to be one for one second of execution on one machine. The cost is calculated using equation 3, where

TABLE I: Comparison our method and previous ones

	Execution time (s)	**Cost**
Using one machine	201	201
Using 11 machines	34	376
Our method	39	266

T is the execution time for k machines and k differs from method to method.

We first executed all steps of the algorithm using one machine (k=1). Then, we executed all steps on the highest number of machines that improves the speedup (which is k=11, according to Figure 5). Finally, we executed the same algorithm considering our approach, where we varied the number of machines we used for each separate execution step, according to the values for each step in Figure 5. Table I contains the execution time for each method along with the calculated costs.

$$cost = \sum_{step_i} T_{step_i}(k) * k \quad (3)$$

It is obvious that our method manages to have an execution time very close to the smallest execution time while having a low cost.

V. Conclusion

We proposed a novel way of utilizing elasticity in cloud computing. Our method steps away from the previous use cases in which, almost uniformly, demand was considered to be "user load". We showed that elasticity can be used to improve execution time and cost of applications that do not have "user load", that is, their demand of resources does not fluctuate in time depending on outside factors. We offered three examples of applications, from very different fields, whose algorithm can be split in multiple distinct steps and we proved that each step requires a different number of resources, depending on a simple policy. By using elasticity, the execution time and cost for these applications can be improved. Our method can easily be applied to many other applications whose execution can take the form of a workflow.

Using this method, we believe that we opened the doors for an entire class of applications that so far were not considered "cloud candidates" to be migrated to the cloud in order to

obtain significant improvements in both cost and execution time.

As future work we intend to extend the method and add boot time for virtual machines as well as the cost to transfer data. We need to identify a better function that indicates the optimal number of machines. We need to use this method with DAG tasks. We need to find a way to run this method online.

ACKNOWLEDGMENT

The research presented in this paper is supported by projects: MobiWay, Mobility beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-PT-PCCA-2013-4-0321; DataWay, Real-time Data Processing Platform for Smart Cities: Making sense of Big Data - PN-II-RU-TE-2014-4-2731. We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. Ieee, 2008, pp. 5–13.

[3] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[4] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 111.

[5] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[6] N. R. Herbst, S. Kounev, and R. H. Reussner, "Elasticity in cloud computing: What it is, and what it is not." in *ICAC*, 2013, pp. 23–27.

[7] G. Galante and L. C. E. de Bona, "A survey on cloud computing elasticity," in *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 263–270.

[8] D. M. Shawky and A. F. Ali, "Defining a measure of cloud computing elasticity," in *Systems and Computer Science (ICSCS), 2012 1st International Conference on*. IEEE, 2012, pp. 1–5.

[9] Y. Guo, M. Ghanem, and R. Han, "Does the cloud need new algorithms? an introduction to elastic algorithms," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 2012, pp. 66–73.

[10] G. Ananthanarayanan, C. Douglas, R. Ramakrishnan, S. Rao, and I. Stoica, "True elasticity in multi-tenant data-intensive compute clusters," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 24.

[11] T. Knauth and C. Fetzer, "Scaling non-elastic applications using virtual machines," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 468–475.

[12] D. Moran, L. M. Vaquero, and F. Galán, "Elastically ruling the cloud: specifying application's behavior in federated clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 89–96.

[13] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 559–570.

[14] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "Sybl: An extensible language for controlling elasticity in cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 112–119.

[15] J. Kuhlenkamp, M. Klems, and O. Röss, "Benchmarking scalability and elasticity of distributed database systems," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1219–1230, 2014.

[16] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 5.

[17] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas, "Cloud elasticity using probabilistic model checking," *arXiv preprint arXiv:1405.4699*, 2014.

[18] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 204–212.

[19] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems*, vol. 32, pp. 82–98, 2014.

[20] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ACM, 2012, pp. 85–96.

[21] E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, "Elasticity in cloud computing: a survey," *annals of telecommunications-annales des télécommunications*, vol. 70, no. 7-8, pp. 289–309, 2015.

[22] H. L. Truong and S. Dustdar, "Programming elasticity in the cloud." *IEEE Computer*, vol. 48, no. 3, pp. 87–90, 2015.

[23] D. L. Eager, J. Zahorjan, and D. Lazowska, "Speedup versus efficiency in parallel systems," *Computers, IEEE Transactions on*, vol. 38, no. 3, pp. 408–423, 1989.

[24] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.

[25] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[26] R. Bellman, "A markovian decision process," DTIC Document, Tech. Rep., 1957.

[27] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, no. 2, pp. 135–164, 2002.