Cloud autoscaling for HTTP/2 workloads

Maria Carla Calzarossa, Luisa Massari*, Momin I.M. Tabash
Department of Electrical, Computer and Biomedical Engineering
Università degli Studi di Pavia
Via Ferrata 5, I-27100 Pavia, Italy
{mcc,luisa.massari}@unipv.it, momin.tabash01@universitadipavia.it

Daniele Tessera
Department of Mathematics and Physics
Università Cattolica del Sacro Cuore
Via Musei 41, I-25121 Brescia, Italy
daniele.tessera@unicatt.it

***Abstract*—Cloud computing provides cost effective solutions for deploying services and applications. Even though resources can be provisioned on demand, they need to adapt quickly and in a seamless way to the workload intensity and characteristics and satisfy at the same time the desired performance levels and the corresponding SLAs. Autoscaling policies are devised for these purposes. In this paper, we apply a state-of-the-art reactive autoscaling policy to assess the effects of deploying the HTTP/2 server push mechanism in cloud environments. Our simulation experiments – that rely on extensions of the CloudSim toolkit – have shown that the autoscaling mechanism is beneficial for web servers even though pushing a large number of objects might lead to server overload.**

Keywords: HTTP/2; Server push mechanism; Cloud computing; Autoscaling policies; CloudSim; Workload characterization; Web workload.

I. Introduction

Cloud computing paradigm is being increasingly exploited in many application domains because of the cost effective scalable solutions provided on demand. In fact, the use of sophisticated virtualization approaches and the flexibility and elasticity of resource provisioning make cloud technologies particularly suitable to cope with workloads characterized by highly variable intensity and resource requirements and by specific performance constraints (see, e.g., [1], [2]). Therefore, to avoid over-provisioning or under-provisioning, it is necessary to adapt automatically and in a timely manner the amount of allocated resources to the workload characteristics.

To cope with these challenging issues, either pro-active or reactive autoscaling policies are being exploited in cloud environments [3]. These policies often take their scaling decisions according to some specific performance indicators periodically measured on the cloud infrastructure as well as on predictions of the resources to be allocated.

In this paper we study cloud autoscaling policies in the framework of web workloads by focusing in particular on the deployment of the HTTP/2 protocol [4]. This new standard protocol promises a significant performance improvement. Nevertheless, the various workarounds (e.g., domain sharding, image spriting, inline resources) – implemented by web masters to cope with HTTP/1.1 inefficiencies – make the transition quite challenging (see, e.g., [5]). It is therefore important to specifically assess the benefits of the new features provided by HTTP/2.

Our paper addresses the so called "server push" optimization mechanism that allows web servers to speculatively send resources without waiting for explicit client requests. This mechanism – first introduced by Google in the SPDY protocol [6] – has good potentials, although its implementation is not straightforward as it requires web masters to identify the bundle of resources to be pushed as well as when the server has to start pushing.

Since no specific best practices nor guidelines have been released so far, we investigate the efficiency of the push mechanism as a function of the workload intensity and web page characteristics. In particular, our study focuses on the web server side and analyzes the effects of autoscaling when server push is being deployed. To the best of our knowledge, this is the first work that analyzes the exploitation of HTTP/2 web servers in cloud environments.

The main contributions of this work are summarized as follows:

- Performance analysis of the server push mechanism in cloud environments;
- Design and development of a simulation environment based on the CloudSim toolkit to exploit a realistic web workload and a state-of-the-art autoscaling policy.

This paper is organized as follows. Section II addresses the state of the art in the framework of web workloads, HTTP/2 and cloud autoscaling policies. Section III presents the evaluation approach proposed for assessing the performance of the server push mechanism in cloud environments. Section IV describes the simulation environment developed to cope with this complex scenario. The simulation results are presented in Section V, while Section VI concludes the paper and outlines some future research directions.

II. Related work

Web workloads have been extensively studied by focusing on aspects, such as page and traffic properties, access patterns and user behavior (see e.g., [7], [8]). Different approaches have been applied to characterize these conventional workloads and derive realistic models. The measurements collected by the web servers in their access logs are typically the basis of these analyses [9], [10].

The characteristics of web pages are another important aspect investigated in the literature. In particular, it has been shown that the increased complexity of the pages – in terms

of number of objects and size – does not significantly affect page load time [11].

Papers investigating the performance of the new features introduced by the HTTP/2 protocol have started recently to appear in the literature. In particular, the performance impact of the server push mechanism has been investigated in the framework of both SPDY and HTTP/2 protocols (see, e.g., [12], [13], [14], [15]). These studies show that page load time in general improves, although an improper use of the push feature could lead to performance degradation. Moreover, while server push is effective on high loss and high latency networks, it can provide little benefits when high speed network connections are deployed.

Despite these papers, we consider the challenges related to the deployment of HTTP/2 server push mechanism in cloud environments. In fact, this mechanism modifies the characteristics of the workload being processed by web servers and the corresponding resource requirements. Therefore, it is necessary to deploy autoscaling mechanisms to dynamically manage cloud resources.

In this framework, an important issue deals with minimizing the amount of allocated resources without violating the QoS constraints. The autoscaling policies proposed in the literature focus on various aspects, such as performance monitoring, workload prediction, adaptivity to dynamically varying workload characteristics and oscillation mitigations (see, e.g., [16] for a detailed taxonomy and survey). Indeed, the tradeoff between the cost of autoscaling – expressed for example in terms of number of autoscaling operations – and its performance benefits has to be accurately taken into account.

The decisions about the allocation or deallocation of cloud resources often rely on some monitored or predicted low-level performance indicators (e.g., utilization of resources, such as CPU, memory, network bandwidth) or high-level ones (e.g., response time, request rate). Metrics that estimate the QoS perceived by users are also proposed in [17] and [18]. The overall objective of these studies is to trigger autoscaling actions as to keep the values of the indicators within some predefined thresholds.

Autoscaling actions can be reactive, i.e., performed as a consequence of some workload changes, or pro-active, i.e., performed according to some predictions of workload changes. Combinations of the two approaches have also been proposed (see, e.g., [19], [20], [21], [22], [23]).

In this paper we do not aim at proposing novel autoscaling policies. We rather apply a state-of-the-art reactive policy to assess its benefits for web servers that implement the server push mechanism.

III. Evaluation approach

Let us recall that a web page typically consists of a base HTML file and multiple embedded objects, e.g., style sheets, JavaScripts, images, each characterized by its own URL. After parsing the HTML file, browsers issue as many HTTP requests as the number of objects referenced within the HTML. In turn, the target web server processes these requests one by one and generates the corresponding HTTP response messages. As already pointed out, HTTP/2 tries to overcome some of the issues typical of this model by allowing the server to push content before it is actually requested by the browser. Therefore, for improving page load time, a single HTTP request can trigger multiple HTTP responses.

For example, to load the home page of the University of Pavia – shown in Fig. 1 – a browser issues 32 HTTP requests. These requests refer to the HTML file and the 31 embedded

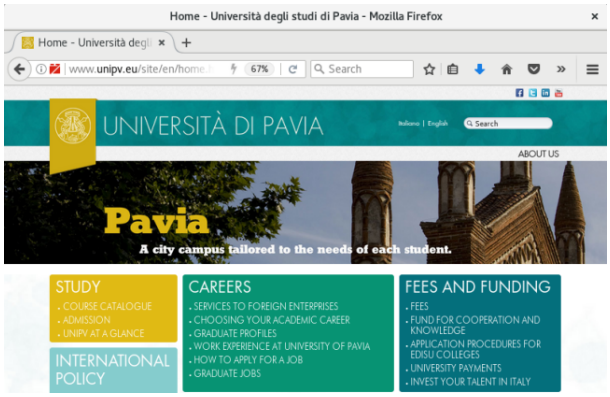

Fig. 1: Snapshot of the home page of the University of Pavia.

objects the page consists of, namely, one style sheet, five JavaScripts and 25 images. With the push mechanism, many of these requests can be saved. In particular, by allowing the server to push images, it is possible to save 25 requests. The images will be sent automatically by the server as HTTP responses. On the contrary, the web server generates responses for the style sheet and the JavaScripts as soon as it receives the corresponding HTTP requests.

Therefore, given the characteristics of the pages of a website – specified in terms of number and types of embedded objects and their size – and the arrival process of the requests – specified in terms of the rate of the HTTP requests received by the server – our evaluation approach considers the problem of provisioning the appropriate amount of cloud resources (i.e., VMs) for the given workload by reacting quickly to any load change, such as peaks of load (e.g., flash crowd) or sudden load decrease.

Autoscaling policies work for this purpose. Among the various policies proposed in the literature, our evaluation relies on a state-of-the-art approach that reacts according to the load conditions of the allocated VMs [24]. In particular, the policy defines two thresholds, that trigger the autoscaling operations. The upper and lower threshold are associated with the maximum and minimum desired utilization of the allocated VMs, respectively. The actual utilization of the VMs is computed on a given number of consecutive time steps.

The evaluation of the benefits of this autoscaling policy on the server push mechanism relies on the simulation of a realistic cloud infrastructure that exploits a web workload.

IV. Simulation environment

For testing the effects of cloud autoscaling policies on the server push mechanism we designed and developed a simulation environment that relies on the CloudSim simulation toolkit [25]. In particular, because of the peculiarities of the scenario under test, we implemented several extensions organized into four modules, namely:

- Web Workload Generator;
- Interactive Broker;
- Autoscaling Policy;
- Monitoring.

Figure 2 summarizes the architecture of our simulation environment and outlines the interactions among the modules and with the core of the CloudSim toolkit.

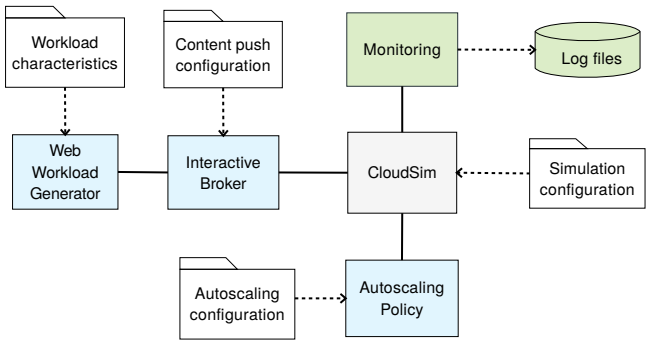

Fig. 2: Architecture of the simulation environment.

As can be seen, CloudSim processes the requests generated by the Web Workload Generator – according to the specified workload characteristics – and scheduled by the Interactive Broker to the VM pool managed by the Autoscaling Policy module. The Monitoring module periodically samples the status of the VMs and collects these performance measurements into log files. Note that the modular architecture of our simulation environment makes it very flexible. This allows for an easy integration of new autoscaling and scheduling policies as well as for customization of the monitoring activities and configuration parameters.

In detail, the *Web Workload Generator* module is responsible of generating the workload, that is, the requests to be processed by the web server. Starting from the properties of the web pages (e.g., number, types and size of the objects) and the arrival process of the HTTP requests (e.g., interarrival times between consecutive requests), the module generates the workload according to the corresponding probability distributions. The Workload Generator relies on the Stochastic Simulation in Java (SSJ) software library[1] to manage the probability distributions associated with the workload properties. In addition, the module supports the generation of multiclass workloads that include web pages with different characteristics as well as with different arrival patterns.

[1]http://simul.iro.umontreal.ca/ssj

Once the requests have been generated, at simulation time the *Interactive Broker* module schedules these requests to the allocated VMs. Various policies, such as last recently used, weighted round robin, and minimum instantaneous load, have been implemented. This module is also responsible of the management of the web content pushing, that is, it pushes objects according to the specified rules. In particular, these rules define the object types to be automatically provided. In addition, the broker is responsible of the allocation and deallocation of the VMs. In detail, when a new VM is allocated it will be ready for processing incoming requests after a given boot time. Moreover, a VM selected for deallocation will be actually deallocated after the completion of the HTTP requests, if any, already scheduled on the VM itself.

The *Autoscaling policy* module implements the reactive autoscaling policy that allocates/deallocates the VMs according to their resource usage evaluated on a given number of time steps. In particular, this module analyzes the utilization of the allocated VMs and triggers allocation/deallocation actions based on their actual load. The properties of the autoscaling policy (e.g., minimum number of VMs that has to be guaranteed) are specified as configuration parameters.

The *Monitoring* module provides a large variety of detailed measurements (e.g., number of VMs allocated/deallocated, VM usage, number of requests being processed, page load time and time spent to load individual objects) to be used for evaluating the performance of the simulated scenario.

V. Experimental results

In this section, we present the experimental results obtained by simulating the scenario depicted in Figure 3 under different HTTP/2 content push configurations.

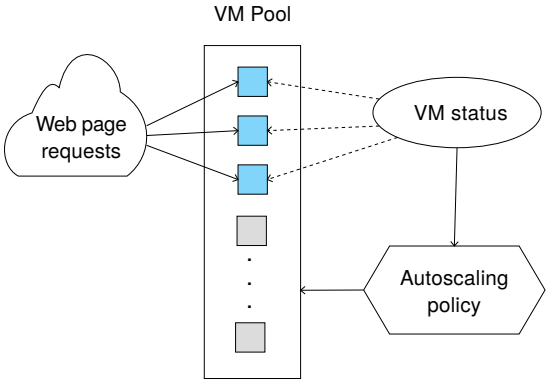

Fig. 3: Experimental scenario.

These experiments are driven by a workload model derived from measurements collected on the web server of the University of Pavia during a 24 hours period. Some 330,000 HTTP requests for web pages were received and processed by the web server. Figure 4 shows their daily arrival pattern as a function of time. As can be seen, the workload intensity is characterized by a clear diurnal pattern, with much fewer

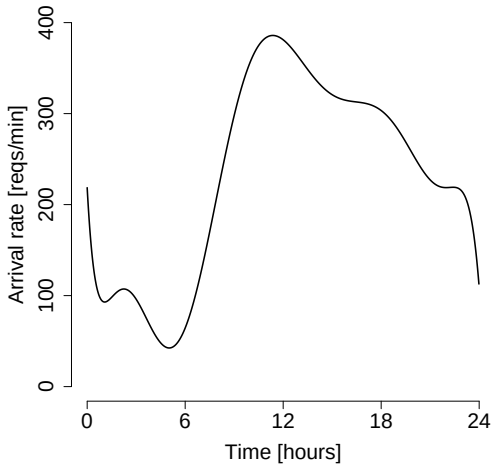

Fig. 4: Daily request arrival pattern.

requests – as little as 42 requests per minute – during the night and early morning hours, and a peak of about 400 requests per minute around noon.

The composition of the web pages, that is, number of embedded objects and their size, is obtained by analyzing the structural properties of the website. On average the web pages reference 40 embedded objects each (see Fig. 1). The processing requirement of each object was set proportional to its size. In detail, we describe the processing time in terms of two equally probable uniform distributions, one in the range $[0.2, 2]$ μs and the other in the range $[110, 118]$ ms.

Another important characteristic of the workload model is the interarrival time between two consecutive requests to the embedded objects of a given page. Since these times are usually quite small, that is, one second or less. We model them in the simulator using two uniform distributions in the range $[0, 1]$ s and $[1, 2]$ s, respectively. The probabilities associated with these distributions are 0.97 and 0.03.

The web workload generated according to these models is then exploited on a simulated cloud infrastructure consisting of a pool of homogeneous VMs (see Fig. 3).

As previously discussed, the reactive autoscaling policy tested in our experiments allocates/deallocates VMs according to their actual utilization. In particular, we carried out several experiments to select the most appropriate values of thresholds that trigger the autoscaling policy as well as the number of time steps that allow the allocated VMs to closely adapt to the workload dynamics, while minimizing the number of scaling operations.

Let us remark that the larger the number of time steps the more conservative the autoscaling policy is. This means that the policy is slow in reacting to sudden changes of load conditions and this can lead to temporary resource under-provisioning or over-provisioning. On the contrary, decisions based on few time steps can easily result in oscillations in the allocated resources. As an example, the step functions of Figure 5 plot the number of allocated VMs for four and ten time steps as a function of time. As can be seen, when decisions are based on fewer time steps (i.e., four), the number of allocated VMs adapts well to the workload intensity.

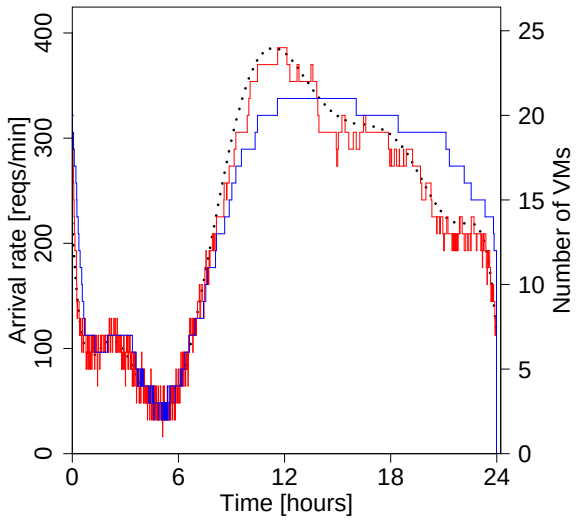

Fig. 5: Behavior of the autoscaling policy with four (red step function) and ten (blue step function) time steps. The dotted curve refers to the simulated daily arrival pattern.

In order to simulate realistic scenarios, we include a 30 seconds boot time for newly allocated VMs. In addition, to avoid oscillations in the number of allocated VMs we include a cooling time, i.e., the minimum time between two autoscaling actions. In our experiment, this time is set to 40 seconds.

Table I summarizes the main parameters used in the simulation experiments.

Different push configurations have been simulated by varying the amount of content to be pushed from 0% (i.e., HTTP/1.1 model) up to 100% (i.e., the entire page is pushed after the request of the HTML file).

The benefits of the server push mechanisms have been evaluated using performance metrics, such as page load time, VM utilization, number of allocated VMs.

Table II presents some statistics of the page load time as a function of the percentage of objects being pushed. As can be seen, the page load time decreases as the percentage of pushed objects increases. Note that in case of "partial" push (i.e., push percentage less than 100%), we assume that for the objects not being pushed the web server generates separate responses as soon as it receives the corresponding HTTP requests.

The benefit of push mechanism is also evident when looking at cumulative distribution function of the page load time. As Figure 6 shows, in case of full push (i.e., 100%) and no push (i.e., 0%), page load times exhibit very different behaviors with much smaller times in the case of full push.

Number of VMs			Boot time	Cooling time	Thresholds		Time steps	Monitoring resolution
Min	Max	Start	[s]	[s]				[s]
1	100	20	30	40	0.4	0.8	4	0.1

TABLE I: Simulation parameters.

	Page Load Time		
Push percentage	Avg.	Median	99-th percentile
0	21.3	21.1	26.9
50	11.2	11.3	17.9
60	9.3	9.2	15.5
70	7.4	7.2	13.6
80	5.5	5.3	12.0
90	3.9	3.0	11.6
100	2.8	2.4	9.1

TABLE II: Statistics of page load time under different push configurations. The times are expressed in seconds.

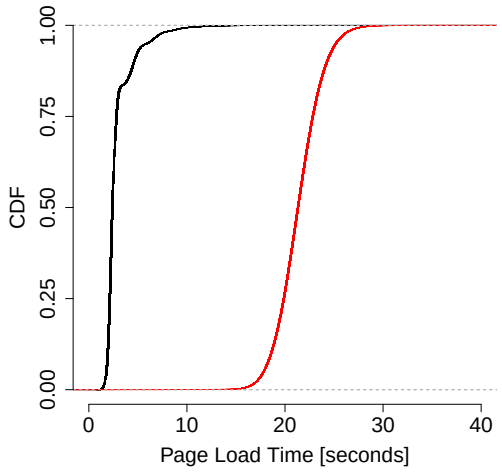

Fig. 6: Cumulative distribution functions of the page load time with full push (black curve) and with no push (red curve).

Page load time has been proven to be a good performance metric for evaluating how the autoscaling policy reacts to load changes. In detail, we analyze in Figure 7 the number of allocated VMs and the number of active VMs as a function of the time of the day. The figure refers to a simulation with full push. The average number of VMs allocated over the 24 hours is 14, while on average only 9 are actively used. In addition, we can notice in Figure 8 that when all allocated VMs are fully utilized, that is, under overload conditions, the page load time significantly increases. Therefore, autoscaling policy quickly reacts by increasing the number of allocated VMs. The page load time then becomes rather stable.

We can also outline that the adopted autoscaling policy leads to rather balanced utilizations, especially with a larger number of allocated VMs. The boxplots of Figure 9 summarize the actual utilization of the VMs as a function of the number of allocated VMs. The diagram refers to a simulation experiment with 80% push. Note that our simulation experiments have

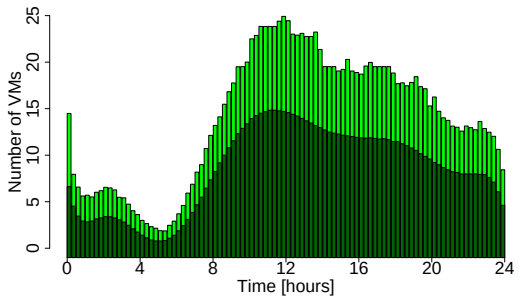

Fig. 7: Number of allocated (light green) and active (dark green) VMs as a function of the time of the day.

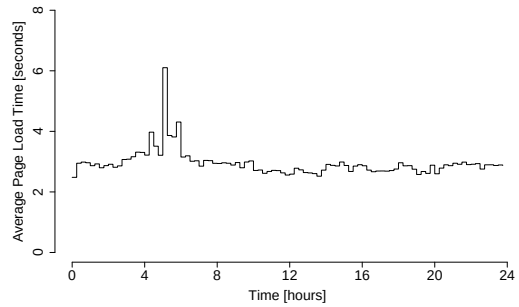

Fig. 8: Average page load time as a function of the time of the day.

shown that this behavior is independent of the fraction of content being pushed.

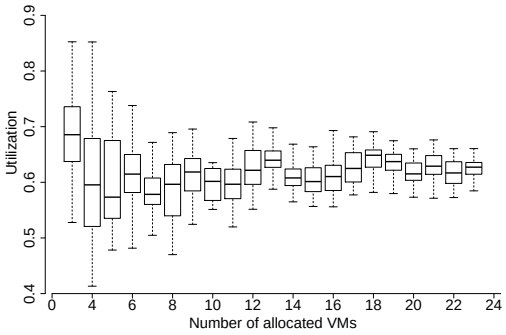

Fig. 9: Boxplots of the actual utilization of the VMs as a function of the number of allocated VMs.

VI. Conclusions

We presented a study aimed at assessing the benefits of the HTTP/2 server push mechanism in cloud environments. A simulation environment based on the CloudSim toolkit has been designed and developed to exploit a web workload on a realistic cloud infrastructure and to implement a state-of-the-art reactive autoscaling policy. The simulation experiments have shown an overall benefit in adopting the push mechanism. In particular, the page load time on the server side is significantly reduced. Moreover, the amount of content being pushed does not influence the behavior of the autoscaling policy.

As a future work, we plan to develop benchmarks to test HTTP/2 push mechanism on a real cloud infrastructure. Moreover, we will study new autoscaling policies and their sensitivity to push configuration parameters.

References

[1] M. C. Calzarossa, M. L. Della Vedova, L. Massari, D. Petcu, M. I. M. Tabash, and D. Tessera, "Workloads in the Clouds," in *Principles of Performance and Reliability Modeling and Evaluation*, ser. Springer Series in Reliability Engineering, L. Fiondella and A. Puliafito, Eds. Springer, 2016, pp. 525–550.

[2] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Computing Surveys*, vol. 48, no. 3, pp. 48:1–48:43, 2016.

[3] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[4] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, 2015.

[5] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is the Web HTTP/2 Yet?" in *Passive and Active Measurement - PAM'16*, ser. Lecture Notes in Computer Science, T. Karagiannis and X. Dimitropoulos, Eds., vol. 9631. Springer, 2016, pp. 218–232.

[6] M. Belshe and R. Peon, "SPDY Protocol-Draft 1," 2012, http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1.

[7] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, 1997.

[8] A. Williams, M. Arlitt, C. Williamson, and K. Barker, "Web Workload Characterization: Ten Years Later," in *Web Content Delivery*, ser. Web Information Systems Engineering and Internet Technologies Book Series, X. Tang, J. Xu, and S. Chanson, Eds. Springer, 2005, vol. 2, pp. 3–21.

[9] M. Calzarossa and L. Massari, "Analysis of Web logs: Challenges and Findings," in *Performance Evaluation of Computer and Communication Systems - Milestones and Future Challenges*, ser. Lecture Notes in Computer Science, K. Hummel, H. Hlavacs, and W. Gansterer, Eds. Springer, 2011, vol. 6821, pp. 227–239.

[10] ——, "Analysis of header usage patterns of HTTP request messages," in *Proc. of the 16th Int. Conf. on High Performance Computing and Communications - HPCC2014*. IEEE Computer Society, 2014, pp. 859–865.

[11] M. Butkiewicz, H. Madhyastha, and V. Sekar, "Understanding website complexity: Measurements, metrics, and implications," in *Proc. of the 11th ACM SIGCOMM Conf. on Internet Measurement - IMC'11*. ACM, 2011, pp. 313–328.

[12] I. N. de Oliveira, P. T. Endo, W. Melo, D. Sadok, and J. Kelner, "Should I Wait or Should I Push? A Performance Analysis of Push Feature in HTTP/2 Connections," in *Proc. of the Workshop on Fostering Latin-American Research in Data Communication Networks - LANCOMM'16*. ACM, 2016, pp. 7–9.

[13] S. Rosen, B. Han, S. Hao, Z. M. Mao, and F. Qian, "Push or Request: An Investigation of HTTP/2 Server Push for Improving Mobile Performance," in *Proc. of the 26th Int. Conf. on World Wide Web - WWW'17*. ACM, 2017, pp. 459–468.

[14] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and A. Wetherall, "How Speedy is SPDY?" in *Proc. of the 11th USENIX Symposium on Networked Systems Design and Implementation - NSDI'14*. USENIX Association, 2014, pp. 387–399.

[15] K. Zarifis, M. Holland, M. Jain, E. Katz-Bassett, and R. Govindan, "Modeling HTTP/2 Speed from HTTP/1 Traces," in *Passive and Active Measurement - PAM'16*, ser. Lecture Notes in Computer Science, T. Karagiannis and X. Dimitropoulos, Eds., vol. 9631. Springer, 2016, pp. 233–247.

[16] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling Web Applications in Clouds: A Taxonomy and Survey," 2016. [Online]. Available: http://arxiv.org/abs/1609.09224

[17] M. D. de Assuncao, C. H. Cardonha, A. S. Netto, and R. L. F. Cunha, "Impact of user patience on auto-scaling resource capacity for cloud services," *Future Generation Computer Systems*, vol. 55, pp. 41–50, 2016.

[18] X. L. Liu, S. M. Yuan, G. H. Luo, H. Y. Huang, and P. Bellavista, "Cloud Resource Management with Turnaround Time Driven Auto-Scaling," *IEEE Access*, vol. 5, pp. 9831–9841, 2017.

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

[20] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and A. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," *ACM Transactions on Computer Systems*, vol. 30, no. 4, pp. 14:1–14:26, 2012.

[21] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling Web Applications in Heterogeneous Cloud Infrastructures," in *Proc. of the IEEE Int. Conf. on Cloud Engineering - IC2E*, 2014, pp. 195–204.

[22] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, Model-driven Autoscaling for Cloud Applications," in *Proc. of the 11th Int. Conf. on Autonomic Computing - ICAC'14*. USENIX Association, 2014, pp. 57–64.

[23] R. Bouabdallah, S. Lajmi, and K. Ghedira, "Use of Reactive and Proactive Elasticity to Adjust Resources Provisioning in the Cloud Provider," in *Proc. of the IEEE 18th Int. Conf. on High Performance Computing and Communications; IEEE 14th Int. Conf. on Smart City; IEEE 2nd Int. Conf. on Data Science and Systems - HPCC/SmartCity/DSS*, 2016, pp. 1155–1162.

[24] M. A. S. Netto, C. H. Cardonha, R. L. F. Cunha, and M. D. de Assuncao, "Evaluating Auto-scaling Strategies for Cloud Computing Environments," in *Proc. of the 22nd Int. Symp. on Modeling, Analysis & Simulation of Computer and Telecommunications Systems - MASCOTS'2014*. IEEE, 2014, pp. 187–196.

[25] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.