Vadara: Predictive Elasticity for Cloud Applications

João Loff, João Garcia
INESC-ID Lisboa, Instituto Superior Técnico - Universidade de Lisboa, Portugal
joao.loff@tecnico.ulisboa.pt, joao.c.garcia@tecnico.ulisboa.pt

***Abstract*—Elasticity is a key feature in cloud computing, and perhaps what distinguishes it from other computing paradigms. Despite the advantages of elasticity, realizing its full potential is hard due to multiple challenges stemming from the need to estimate workload demand. A desirable solution would require predicting system workload and allocating resources *a priori*, i.e., a *predictive* approach. Instead, what is mainly available are *reactive* solutions, requiring difficult parameter tuning.**

Since each *Cloud Provider* (CP) has its own implementation idiosyncrasies, it's impossible for developers to: (i) learn only about one platform and re-use that knowledge in others; (ii) migrate developed elasticity solutions between different CPs; and (iii) to develop reusable predictive elasticity rules or algorithms.

This paper makes three contributions to provide an effective elasticity environment. First, *Vadara*, a totally generic elasticity framework, that transparently connects and abstracts several CPs API behaviour, and enables the use of pluggable CP-agnostic elasticity strategies.

Second, it presents a predictive workload forecasting approach, which ensembles several individual forecasting methods, and introduces a padding system based on the most recent prediction errors for both under- and over-provisioning.

Finally, results show (1) Vadara's successful connection to well-known CPs, (2) the improvements made regarding under- and over-provisioning due to our padding system, and (3) the effectiveness of our ensemble forecasting technique.

***Keywords*-cloud computing; elasticity; demand forecasting; ensemble forecasting; cloud monitoring**

I. Introduction

One of the key goals of cloud computing is the elasticity of computational resources. Users expect CPs to provide any quantity of resources on short notice. In particular, it's expected that the resources can be (1) provisioned when an application load increases (scale up) and (2) released when load decreases (scale down) [1]. A system's ability to adapt to workload changes by allocating and deallocating resources, such that at each moment the available resources match the current demand as closely as possible, is called *elasticity* [2].

Despite the perceived advantages of elasticity, realizing its full potential is hard due to multiple challenges stemming from the need to precisely estimate resource usage in the face of significant variability in workload patterns. A desirable solution would require an ability to predict the incoming workload on the system and allocate resources a priori. This capability in turn will enable the application to be ready to handle the load increase when it actually occurs [3].

Instead, what current CPs offer are *reactive-based* solutions (e.g. [4]). In this type of system, a given threshold or rule has to be breached before the system reacts and changes [5], [6]. These solutions require expert and tedious human development and maintenance interventions. Fine tuning for the specific demand pattern is necessary for it to yield better results, which is often difficult to figure out [3], [7].

Current cloud computing APIs have not been the subject of active standardization [5]. Thus, customers can't easily migrate their data and programs from one provider to another. This issue is known as CP lock-in, one of the key issues identified as holding back the expansion of cloud computing [1], [8]. The obvious solution is to standardize the APIs so that any cloud application user could deploy services and data across multiple CPs [5].

Moreover, users should be allowed, using the most common CP monitoring and scaling interfaces, to design customized elasticity algorithms that can be plugged into any existing CP without additional tinkering [9].

Vadara is a new, totally generic framework regarding the employed elasticity strategy; that bypasses CP elasticity lock-in, allowing the development of elasticity strategies that are not tied with any CP; that abstracts the CPs APIs and design specificities thereby decoupling elasticity from any underlying CP; that allows a cloud application to be deployed at multiple CPs at the same time; and that takes full-period and instance startup time issues into consideration.

In addition to the framework, we also present results from deploying Vadara on several CPs using various elasticity strategies that forecast the future workload demand based on metrics retrieved from CPs monitoring APIs. This type of approach, called *predictive*, uses heuristics and analytical techniques to anticipate the system load, and based on these results, decides when and how to scale resources [5].

Our predictive approach uses a new padding technique for several individual forecasting methods, based on the most recent prediction errors for under- and over-provisioning. Those individual methods are then combined in a single forecasting technique, that uses a weighted *k-Nearest Neighbors* (kNN) algorithm.

In short, this paper's contributions are:

- Proposing Vadara a totally generic framework regarding the employed elasticity strategy that allows the development of CP agnostic elasticity approaches.
- Introducing a padding technique for demand forecasts that takes into account recent prediction errors and past occurrences of under- and over-provisioning, for a variety of individual forecasting methods.
- Advancing a new ensemble forecasting algorithm, a *k-Nearest Neighbors* (kNN) algorithm weighted by recent forecast performance.

- Describing a series of experiments that test our proposed forecasting techniques, using trace data from Google Cluster Data [10]. The evaluation results demonstrate that our methods are effective.

This paper continues with the definition of cloud elasticity challenges (Sec. II). We follow that with a description of Vadara's architecture (Sec. III), and of the methodology for our forecasting techniques (Sec. IV). This is succeeded by the results of Vadara's evaluation using a variety of forecasting techniques, including our proposed forecasting technique (Sec. V). We conclude with an overview of the current cloud elasticity ecosystem (Sec. VI), and some final remarks (Sec. VII).

II. Cloud Elasticity Challenges and Issues

This section discusses the challenges to realizing elastic resource provisioning for cloud-based applications.

A. Cloud interoperability

A possible solution to the resource availability problem is the use of multiple clouds to ensure the required amount of resources. According to Galante et al. [5] even though there's some academic work regarding combining local and public clouds, the combined use of different public clouds remains challenging. The reason for the current poor portability and limited interoperability between clouds is the lack of standardized API's. Consequently, each cloud provider has its own way of interacting with cloud clients/applications/users [8].

B. Startup time

One important fact in the cloud elasticity process is that although cloud users can make their acquisition requests at any time, it may take some time for the acquired resources to be ready to use. This is called *startup time* [11], [12]. In a perfectly elastic cloud, there would be no time delay between detecting load changes and changing resourcing levels. However, in real world clouds, the startup time can vary (as seen in [13]). Thus, resources provisioning could be slower than expected, affecting the efficacy and efficiency of elasticity mechanisms [5].

C. Full-period

Imagine a developer has already paid for a full rental period (e.g. one hour) of a computing instance and orders it to shutdown before the rental ends. In this situation there's no need to shut it down before the full-rental period is complete. This issue is known as the *full-period* issue and a reasonable policy is that whenever an instance is started, it is better to shut it down only when approaching full period operation [12].

D. Workload forecasting

Realizing the full potential of cloud elasticity is hard due to multiple challenges stemming from the need to precisely estimate resource usage in the face of significant variability in client workload patterns. It's desirable that the resources can be acquired earlier than the time when workload actually increases [14]. This outcome can only be possible if the future workload can be predicted, possibly using historical data.

III. Vadara's Architecture

In this section we start by defining the requirements for building a framework such as Vadara. We then present its architecture and typical workflow.

A. Requirements

From the defined challenges (Section II), we've define Vadara's requirements:

- It has to work with any CP. Extensions to connect with any of the providers should be developed and easily plugged into the framework. It has to be extensible and decoupled from any CP, abstracting the underlying infrastructure.
- It should allow the development of totally generic elasticity strategies and the resulting modules should be seamlessly pluggable into the framework. Furthermore, connectors between Vadara and the elasticity approach should be simple and clear.
- It should work with multiple CPs simultaneously. The framework should generalize the interaction between Vadara and the CPs, thereby standardizing all CPs APIs.
- It should store information regarding instances launch and startup times, enabling the framework to track CPs startup times, and instances full-period status. Hence, a data repository specialized in time-series should be available within the framework.

B. Architecture

On the left side of Fig. 1, we can see a typical CP platform with the computing nodes where applications run. Vadara assumes that the application running at the CP is a cloud application that can seamlessly scale out. We have two core cloud services[1] that act upon those nodes: the *scalability service*, responsible for executing scaling commands over the node group (i.e. adding or deleting nodes); and the *monitoring service*, responsible for answering requests for performance metrics from the existing nodes (e.g. current CPU load). In order to connect to Vadara, a CP only has to provide APIs for monitoring and scaling, both common in today's CPs (e.g. [4], [15]).

On the right of Fig. 1 we have Vadara's four main components and a fifth data storage component:

***core*:** it coordinates the initialization of all Vadara's components. It's responsible for correctly configuring and initializing all the other components, including the *CP extensions*.

***decider*:** it's responsible for pondering which elasticity actions to take. It starts by requesting the *monitor* instance metrics. Upon receiving those metrics it analyses them, taking the appropriate action. That decision is then communicated to the *scaler*.

***monitor*:** it's the bridge between the *decider* and the CPs *monitoring services*. It's responsible for forwarding the

[1] We decided to call services to actions that CPs make available to users (usually through an API).

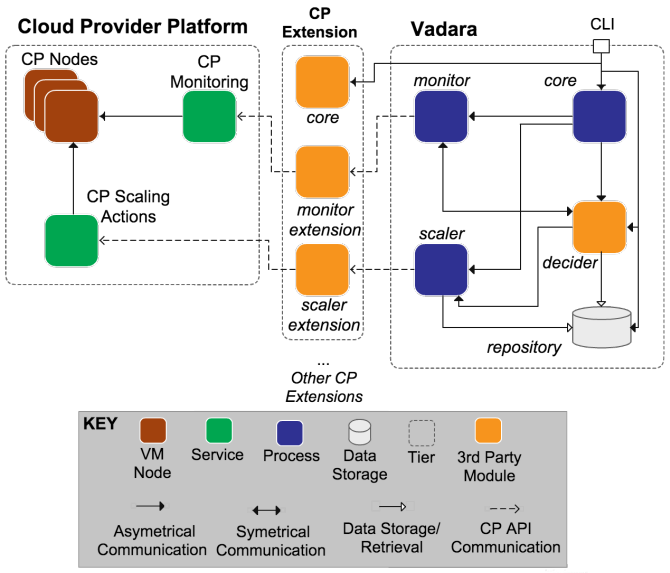


Fig. 1. Vadara's Architecture

decider's metric requests to the various CPs *monitoring services*. Once it has all the CPs replies, it aggregates them and replies to the *decider*.

***scaler*:** it's the bridge between the *decider* and the CPs *scalability services*. It's responsible for forwarding the *decider's* scaling requests to the various CPs *scalability services*. Once is has all the CPs replies, it aggregates them and replies to the *decider*. Furthermore, it's responsible for keeping track of each instance's full-period status and startup times.

***repository*:** it's responsible for persistently storing any information required by other components (e.g., metrics received, CP instance startup time).

Between Vadara and the CP platforms, we have the *CP extensions*, which act as adapters between Vadara and each CP. Each of these extensions has three components, similar to the ones we find in Vadara. The *extension*'s *core* is responsible for initializing all *extension* components. The *extension*'s *monitor* is responsible for translating between Vadara's and CP metric definitions, and to perform the correct CP monitor API call. The *extension*'s *scaler* is responsible for translating between Vadara's and CP scaling concepts, and then performing the correct CP scaling API call.

In order to achieve API standardization, we created a common terminology to be used within Vadara's components. This terminology, standardizes CPs *services*, enabling *deciders* to use the same API call for different CPs. At the same time, Vadara allows *deciders* to use CP specific API calls and concepts, allowing developers to build *deciders* that run within Vadara but interface only with a specific CP.

C. Execution Workflow

Vadara operates on a simple three step loop. First, the *decider* asks the *monitor* for metric data. The *monitor* forwards that request to the available *CP extensions*, which in turn forward it to the corresponding CP. Upon receiving the answers, the *monitor* aggregates them, replying back to the *decider*. Second, and using that reply and additional historical data, the *decider* chooses an action to take accordingly to its defined logic (e.g. a forecasting algorithm). Finally, the *decider* passes that decision onto the *scaler*. The *scaler* then forwards it to the available *CP extensions*, which in turn forward it to its CPs.

Note that the *decider* can store in the *repository* any data it wants. This information can be invaluable to improve the *decider*'s decisions. Also stored into the *repository* is information regarding each instance launch time, used to track its full-hour period, and instances startup times. This behavior is built into the *scaler*, so each *CP extension* does not need to re-implement this.

IV. Workload Forecasting Methodology

Workload forecasting techniques allow elasticity solutions to predict the future system load and allocate resources *a priori*. This enables applications to handle the load increase when it actually occurs. The best capacity predictor should be the one that minimizes the sum of errors over multiple forecasts, specially under-provisioning errors since they may cause significant *Service Level Agreement* (SLA) violations.

We first detail our padding technique for individual forecasting methods, and then we explain our ensemble forecasting approach. Lastly, we discuss forecast accuracy measures.

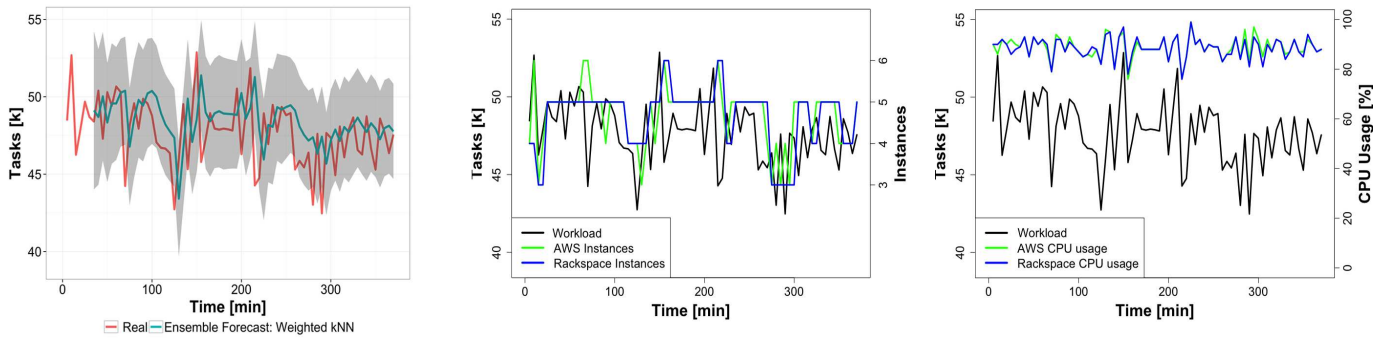
A. Individual forecasting methods

For each one-step-ahead forecast we observe under-provisioning (U_t) and over-provisioning (O_t) situations, counting their occurrences (n_U and n_O respectively), and recalculate their *Exponential Mean Error* (EME) ($EME_t(U_t)$ and $EME_t(O_t)$). EME is a weighted average based on simple exponential smoothing, where weights exponentially increase over time, i.e. most recent error observations have more weight than older ones. It's given by,

$$EME_t(Y_t) = \alpha Y_t + (1-\alpha) EME_{t-1}(Y_{t-1}), 0 \leq \alpha \leq 1, \quad (1)$$

where Y_t is the new observed value at time t, EME_{t-1} is the previous EME at time $t-1$, and α is a smoothing factor optimized minimizing the sum of squared one-step forecast errors. If $\hat{Y}_t$ is the prediction of Y_t then $e_t = Y_t - \hat{Y}_t$ is the one-step forecast error, hence, we aim to minimize $\sum e_t^2$. Just like in Jiang et al. [14] approach, we are actually splitting our time-series into under- and over-provisioning components, to better analyze contributions from each one.

Adopting an approach based on Shen et al. [16], we calculate a *padding* value to add to each forecast. This padding aims to avoid under-provisioning errors by adding a small extra value to the predicted resource demand. Our approach is based on the observation that under-provisioning errors are



(a) (b) (c)

Fig. 2. One the left (Fig. 2(b)) we have our ensemble forecasting method using Google Cluster Data trace. Grey area represents a forecast confidence interval of 80%. For the same run, we have on the middle (Fig. 2(b)) the number of instances for AWS and Rackspace initiated through Vadara, and on the right (Fig. 2(c)) CPU usage.

often caused by resource usage bursts. Thus, we choose the padding value based on the recent burstiness of application resource usage and recent prediction errors. Hence, we use EME (Equation 1) to measure errors from previous forecasts. The padding value is a weighted average between observed under- and over-provisioning occurrences,

$$pad_t = \frac{n_O}{n} EME_t(O_t) + \frac{n_U}{n} EME_t(U_t), \qquad (2)$$

where n is the total number of previous observations. Note that since over-provisioning errors are negative, pad_t can also be negative if over-provisioning contribution is bigger than under-provisioning. Finally the forecast at time t can be given by $\hat{Y}_t = pad_t + \hat{v}_p(t)$, where $\hat{v}_p(t)$ is the predicted value given by a specific individual forecasting method p for a time t.

B. Ensemble forecasting methods

To better capture the temporal dynamics of the workload demand, we propose the combination of multiple forecasting methods for two reasons: (1) the robustness of the ensemble method mitigates the risk of large deviation for prediction results, and (2) the ensemble method is on average better than an individual predictor [14], [17]–[19].

Literature [18], [19] shows that when there is much uncertainty in finding the best model, like in the present work, combining may improve prediction accuracy. Further, according to Armstrong [20], combining is more useful for short range forecasting, like in the present work, where random errors are more significant.

Our technique for combining forecasting methods is based on a weighted *k-Nearest Neighbors* (kNN) algorithm [21], which outputs a weighted average of the k methods with the lowest error value for recent forecasts. The weight for each individual method is equal to the inverse of the recent forecast performance. Recent forecast performance is measured by applying EME to the individual method's forecast accuracy values (A_t). The forecast value for the ensemble approach is given by,

$$\hat{Y}_t = \sum_{i=1}^{k} w_i Y_{p_t}, \text{with } w_i = {}^{1}/_{EME_t(A_{t_p})}, \qquad (3)$$

where k is $\sqrt{n}$ (Duda et al. [21]) and n is the number of individual methods used. Y_{p_t} is the forecast from an individual forecasting method p. $EME_p(A_{t_p})$ is the EME of accuracy values A_t for the same individual method p.

C. Forecast Accuracy measure

There is no universally preferred measure of accuracy estimation in forecasting, therefore experts often disagree as to which measure should be used [22]. We've selected *Mean Absolute Percentage Error* (MAPE) to measure accuracy in our work, as it's widely used in cases of combining and selecting forecasts [18], [19].

V. Evaluation

We aimed to evaluate Vadara's ability to correctly and timely control elasticity behavior. We evaluated how Vadara adjusts a cloud-based application, deployed on both AWS and Rackspace, in order to maintain a stable CPU load on the computing nodes while saving the cost of additional computing nodes whenever possible.

AWS ran 't2.micro' nodes with 1 vCPU and 1GB of RAM memory. Rackspace ran '512MB Standard Instance' with 1 vCPU and 512MB of RAM memory. Both providers also ran a load-balancer to distribute the requests amongst all the nodes, using a round-robin approach.

We used *TraceVersion1* of the Google Cluster Data trace [10]. Other authors [6] concur that this is a generic trace that includes both medium and short term burstiness patterns. The workload consists of a sequence of web requests, that occur at a 5 minute interval timestamp, over a 7 hour period. The number of requests was aggregated by timestamp.

We used a selection of common forecasting algorithms and also our ensemble forecasting algorithm, to demonstrate that Vadara transparently manages cloud applications' elasticity.

A. Individual Forecasting Methods

Three individual methods were chosen to analyze our padding system: Holt-Winters, ARIMA and StructTS. These are known methods used to perform time-series forecasting in cloud elasticity scenarios [3], [6], [7], [14].

The addition of padding to the standard forecasting lowered observed under-provisioning occurrences from 50% to 22% in Holt-Winters, from 46% to 13% in ARIMA, and from 54% to 18% in StructTS. Observed over-provisioning raised at the same rate, from 50% in Holt-Winters, 54% in ARIMA and 46% in StructTS to 78%, 87% and 82% respectively.

B. Ensemble approach: kNN Weighted Average

In Fig. 2(a) we applied our ensemble forecasting approach by combining Holt-Winters, ARIMA, and StructTS, all using our padding technique. Observed under-provisioning occurrences increased relatively to individual approaches, near 12%. Meanwhile, over-provisioning situations stayed near 65%. With this approach we also got an increase in near perfect forecast situation, with over 13%.

However, under- and over-provisioning are not the litmus test for forecasting since big and small errors become indistinguishable within those metrics. In terms of MAPE, a more clarifying metric in this case, the ensemble approach has a MAPE of 2.5%, a staggering improvement over any of the individual techniques: 5.7% for Holt Winters, 4.7% for ARIMA, and 4.3% for StructTS.

Finally, by plugging in our ensemble forecaster as the Vadara *decider* and injecting the trace onto our application, we can see (Fig. 2(b)) that, due to demand being closely forecast, the total computing nodes requested by the *decider* follows the pattern of the real demand. AWS nodes respond faster to changes in workload due to a faster instance startup time, meanwhile, due to a slower startup time, Rackspace exhibits longer periods of under-provisioning.

Moreover, we kept resources maximized throughout the trace's duration, since CPU utilization averaged near 89% with a standard deviation of less than 5% (2(c)), an even more significant result due to the trace's high burstiness degree.

VI. Related Work

For a comprehensive study regarding the state of the art of elasticity in the cloud, in both commercial and academic solutions, readers are referred to Galante et al. [5] and Lorido-Botrán et al. [6]. We highlight the most prominent elasticity-related solutions regarding (1) reactive approaches, (2) predictive approaches, and (3) frameworks that support those approaches.

A. Reactive approaches

Reactive, or threshold-based, solutions typically require the user to specify the threshold values on the resource usage (e.g., requests per second). Those thresholds, when reached, trigger actions on the underlying cloud. Such rule-based approaches depend on manual tuning to the specific demand pattern, to yield better results. However, it is often difficult for users to figure out the proper scaling conditions. By contrast, our approach does not require the user to specify any scaling rules.

The use of reactive techniques is quite common and is found in most commercial solutions, such as provided by *Amazon Web Services* (AWS) [4], RightScale [23], Scalr [24], and Enstratius [25]. Vaquero et al. [26] present a comprehensive survey on reactive-based academic solutions, discussing works focused on control theory applied to reactive solutions, and also on the standardization of user-defined rules.

B. Predictive approaches

Predictive approaches use heuristics and analytical techniques together with historical data to predict future demand and proactively allocate resources. Apart from AzureWatch [27], which works exclusively with Microsoft Azure, we are not aware of any commercial solution that uses a predictive elasticity mechanism.

PRESS [28] and CloudScale [16] employ Fast Fourier Transform (FFT) to identify repeating patterns, and use a discrete-time Markov chain to predict demand for the near future. Both approaches are focused on predicting individual *virtual machine* (VM) resource demand. In our work we predict demand for the whole cloud deployment.

Roy et al. [3] employs an autoregressive-moving-average model (ARMA), which predicts future workload based on a limited workload history. The ARMA model parameter choice was optimized for the specific trace used, hindering the use of this approach for different traces.

Jiang et al. [14] decomposes the cloud capacity into provisioning and de-provisioning components, estimating each one independently. Gandhi et. al. [7] employs Kalman filters to automatically learn the (possibly changing) system parameters for each application. Both approaches present a very high number occurrences of under-provisioning, which as we saw, is critical for cloud customers.

Saripalli et al. [29] use a two-step approach, using cubic spline interpolation combined with a hotspot detection algorithm for sudden spikes. Predicted values depend on the window width chosen. Hence, large amounts of available data are needed to produce confident predictions.

C. Cloud elasticity frameworks

Most commercial frameworks don't provide a way for customers to input their own elasticity rules and methods, disabling them from building more complex and custom-tailored approaches. Both Enstratius [25] and Scalr [24] allow users to create custom rules, but only reactive ones.

Yang et al. [9], Krannas et al. [30] and Morais et al. [31] propose frameworks that cope with cloud elasticity, but none provide a feature set similar to Vadara's. All are unable to work with multiple CPs at the same time, and there's no concern over full-period and startup time issues. Furthermore, in [9], it's not possible to plugin a user specific elasticity strategy. Meanwhile, Mao et al. [12] considers full-period and startup time issues, but does not offer a generic framework that solves those issues regardless of the approach used.

VII. Conclusions

Despite the perceived advantages of elasticity, realizing its full potential is hard due to multiple challenges stemming from the need to precisely estimate resource usage. A desirable solution would require an ability to predict the incoming workload and allocate resources *a priori*, i.e. a *predictive* approach. Instead, what current CPs offer are *reactive-based* solutions, requiring often difficult fine tuning for the specific demand pattern, for them to yield better results.

We presented Vadara, an elasticity framework which enables the creation of elasticity strategy modules that are totally generic and pluggable into the framework. We showed that (1) it's highly generic and extensible to work with any CPs (even private ones), (2) can handle multiple CP at the same time, (3) supports multiple modular elasticity strategies, and (4) takes into consideration full-period and startup time problems.

Moreover, we proposed an effective ensemble method to predict future workload demand, based on a weighted kNN-like approach that chooses the methods that were recently closer to the actual demand. This approach relies on individual forecasting techniques, that were padded with an over- and under-provisioning aware methodology. With this approach, when comparing with the typical forecasting algorithms, we were able to reduce under-provisioning observations by over 12%, to increase accurate forecasts by nearly 13%, and to reduce MAPE by more than half.

Acknowledgments: This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under project PEst-OE/EEI/LA0021/2013.

References

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

[2] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 23–27. [Online]. Available: https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst

[3] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 7 2011, pp. 500–507.

[4] Amazon, "Amazon web services," 2014, last visited: 2014/08/14. [Online]. Available: aws.amazon.com

[5] G. Galante and L. C. E. D. Bona, "A survey on cloud computing elasticity," in *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, no. 1. IEEE, 11 2012, pp. 263–270.

[6] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling techniques for elastic applications in cloud environments," Department of Computer Architecture and Technology, UPV/EHU, Tech. Rep., 2012. [Online]. Available: http://www.sc.ehu.es/ccwbayes/isg/index.php?option=com_jresearch&view=publication&task=show&id=780&Itemid=89

[7] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, model-driven autoscaling for cloud applications," in *11th International Conference on Autonomic Computing (ICAC 14)*. Philadelphia, PA: USENIX Association, 2014, pp. 57–64. [Online]. Available: https://www.usenix.org/conference/icac14/technical-sessions/presentation/gandhi

[8] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. IEEE, 2010, pp. 27–33.

[9] J. Yang, J. Qiu, and Y. Li, "A profile-based approach to just-in-time scalability for cloud applications," in *2009 IEEE International Conference on Cloud Computing*. IEEE, 2009, pp. 9–16.

[10] J. L. Hellerstein, "Google cluster data," 2010, last visited: 2014/08/14. [Online]. Available: http://googleresearch.blogspot.com/2010/01/google-cluster-data.html

[11] P. C. Brebner, "Is your cloud elastic enough?" in *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12*. New York, New York, USA: ACM Press, 2012, p. 263.

[12] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 5 2013, pp. 67–78.

[13] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 6 2012, pp. 423–430.

[14] Y. Jiang, C.-s. Perng, T. Li, and R. Chang, "Self-adaptive cloud capacity planning," in *2012 IEEE Ninth International Conference on Services Computing*. IEEE, 6 2012, pp. 73–80.

[15] Rackspace, "Rackspace," 2014, last visited: 2014/08/14. [Online]. Available: www.rackspace.com

[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 - SOCC '11*. New York, New York, USA: ACM Press, 10 2011, pp. 1–14.

[17] R. Adhikari and R. K. Agrawal, "Combining multiple time series models through a robust weighted mechanism," in *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*. IEEE, 3 2012, pp. 455–460.

[18] C. Christodoulos, C. Michalakelis, and D. Varoutas, "Forecasting with limited data: Combining arima and diffusion models," *Technological Forecasting and Social Change*, vol. 77, no. 4, pp. 558–565, 5 2010.

[19] H. Zou and Y. Yang, "Combining time series models for forecasting," *International Journal of Forecasting*, vol. 20, no. 1, pp. 69–84, 1 2004.

[20] J. Armstrong, "Combining forecasts: The end of the beginning or the beginning of the end?" *International Journal of Forecasting*, vol. 5, no. 4, pp. 585–588, 1 1989.

[21] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. Wiley-Interscience, 2000.

[22] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 10 2006.

[23] RightScale, "Rightscale," 2014, last visited: 2014/08/14. [Online]. Available: http://www.rightscale.com/

[24] Scalr, "Scalr," 2014, last visited: 2014/08/14. [Online]. Available: http://www.scalr.com

[25] Enstratius, "Enstratius," 2014, last visited: 2014/08/14. [Online]. Available: www.enstratius.com

[26] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, p. 45, 1 2011.

[27] AzureWatch, "Azurewatch," 2014, last visited: 2014/08/14. [Online]. Available: http://www.paraleap.com/azurewatch

[28] Z. Gong, J. Wilkes, and X. Gu, "Press: Predictive elastic resource scaling for cloud systems," in *2010 International Conference on Network and Service Management*. IEEE, 10 2010, pp. 9–16.

[29] P. Saripalli, G. Kiran, R. R. Shankar, H. Narware, and N. Bindal, "Load prediction and hot spot detection models for autonomic cloud computing," *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pp. 397–402, 12 2011.

[30] P. Kranas, V. Anagnostopoulos, A. Menychtas, and T. Varvarigou, "Elaas: An innovative elasticity as a service framework for dynamic management across the cloud stack layers," in *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*. IEEE, 7 2012, pp. 1042–1049.

[31] F. J. A. Morais, F. V. Brasileiro, R. V. Lopes, R. A. Santos, W. Satterfield, and L. Rosa, "Autoflex: Service agnostic auto-scaling framework for iaas deployment models," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 5 2013, pp. 42–49.