2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models

Fábio Morais*, Francisco Brasileiro*, Raquel Lopes*, Ricardo Araújo*, Wade Satterfield†, Leandro Rosa‡
* *Universidade Federal de Campina Grande – Systems and Computing Department – Campina Grande, PB – Brazil*
Email: fabio@lsd.ufcg.edu.br, fubica@dsc.ufcg.edu.br, raquel@dsc.ufcg.edu.br, ricardo@lsd.ufcg.edu.br
† *Hewlett-Packard – ESSN – Fort Collins, Colorado – USA*
Email: Wade.Satterfield@hp.com
‡ *Hewlett-Packard – ESSN Brazil Lab – Porto Alegre, RS – Brazil*
Email: Leandro.Rosa@hp.com

***Abstract*—Elasticity is a key property to reduce the costs associated with running services in cloud systems that employ an infrastructure-as-a-service (IaaS) deployment model. However, to be able to exploit this property, users of IaaS systems need to be able to anticipate the short-term future demand of their own services, so that only the required infrastructure is requested at any instant in time. This guarantees that service level objectives (SLOs) are always honored by the users of IaaS systems, while over provisioning is avoided. The process of automatically change the amount of resources used to run a service in an IaaS system is named auto-scaling, and the state-of-the-practice uses simple reactive approaches. Although these approaches can successfully reduce the costs due to over provisioning, they are frequently insufficient to minimize the costs due to SLO violations. For that, proactive approaches are required. In this paper we propose a framework for the implementation of auto-scaling services that follows both reactive and proactive approaches. The latter is based on the use of a set of predictors of the future demand of services deployed over IaaS resources, and a selection mechanism that chooses, over time, what is the best predictor to be used. We have also proposed a correction method that uses historical data on the predictors' errors to reduce the probability of under provisioning the services, thus further diminishing the number of SLO violations. We have evaluated the performance of the proposed approach using production traces of HP customers. Our results show that costs savings of as much as 37% can be achieved, while the probability of an SLO violation can be kept, on average, as small as 0.008%, and never larger than 0.036%.**

***Keywords*-auto-scaling; capacity management; workload prediction; multiple predictions; infrastructure-as-a-service.**

I. Introduction

For some time now, more and more companies are adopting the *infrastructure-as-a-service* (IaaS) deployment model of cloud computing as their primary way of provisioning computing capacity. This can be either based on a private IaaS deployment model, where the cloud paradigm is used to reduce the costs of running the IT infrastructure by exploiting economy of scales in the in-house infrastructure, or based on a public IaaS deployment model, where capacity is acquired on-demand from external IaaS providers, which normally charge the usage of their infrastructure following a pay-as-you-go pricing model.

Many of the services that need to be migrated to the cloud are critical to the business of the companies that run them and present time-varying demands. Indeed, *elasticity* is one of the main advantages offered by IaaS providers to their users. An optimal strategy to run services with time-varying demands over IaaS resources would be one that could allocate, at any moment in time, exactly the capacity that is needed to keep the services running with the required performance, i.e. satisfying their service level agreements (SLAs). Such approach would *automatically scale* the capacity provisioned, allocating more resources when demand increases, and releasing resources as soon as they are no longer needed.

Clearly, an important aspect of an efficient auto-scaling mechanism is the ability to anticipate service demand changes. In this regard, monitoring and prediction are important tasks that need to be carefully accomplished by the auto-scaling mechanism. However, anticipating demand is a complex task that has been neglected by short-term capacity planning solutions, which only react to load changes [1], [2], [3], or consider that a predictor will be available [4], [5], without really assessing the influence of these predictors in the performance of the auto-scaling mechanism.

Moreover, some of the previous works require that service information such as response times, queue lengths, arrival rates and request demands are gathered [4], [5], [2], which raises privacy issues to the service providers, since these are sensitive information that they may not be willing to share, even with their IaaS providers.

In this paper we address the limitations of prior work by proposing a hybrid proactive-reactive auto-scaling mechanism that is: *service agnostic* – it only needs to monitor the resource utilization of the infrastructure where the services run, i.e. it is a non-intrusive framework; and *dynamically and automatically configured* with respect to the prediction and monitoring tasks. In particular, it considers the use of multiple predictors, since there is no predictor that is good to anticipate load for all services [6]. It continually assesses prediction options that are available and judiciously chooses the predictor to be used in the near future, allowing the mechanism to adapt itself to changes on load patterns. In addition, we propose a new prediction correction method

978-0-7695-4996-5/13 $26.00 © 2013 IEEE
DOI 10.1109/CCGrid.2013.74

that aims at avoiding underestimations errors, a need that was also perceived by other researchers in the resource management context [7], contributing to further decrease the number of SLA violations.

The rest of this paper is structured as follows. We start by reviewing the related literature concerning automated capacity planning in the context of virtualized infrastructures (Section II). Then, we give a clear description of the problem we are addressing, including the assumptions that are made (Section III). This is followed by the presentation of a framework for the implementation of service agnostic auto-scaling mechanisms (Section IV). We then instantiate the proposed framework and evaluate the performance of the resulting auto-scaling mechanism (Section V). We conclude the paper with our final remarks and directions for future research.

II. Related Work

Resource management of virtualized infrastructures has been addressed by many researchers, whose work takes into account one or more of the following activities:

- Long-term capacity planning: decide how many resources are needed to run the infrastructure in the long term (at least one year ahead) [8];
- Mid-term capacity planning: decide how many physical resources to turn on in the next couple of hours or day, mainly with the purpose of reducing energy costs [6], [9];
- Short-term capacity planning: decide how many virtual machines (VMs) to deploy to a given application, or service, in the next short-term interval, which is in the order of minutes, and usually no longer than one hour [3], [4], [1], [5].
- VM placement: decide where to create the VMs that are needed on top of the available physical resources, trying to maximize resource usage and reduce costs, while not compromising performance [10], [11], [12];

All these areas are strongly related. The short-term capacity manager needs to instantiate VMs on top of the physical resources, which is done by the VM placement manager. The VM placement manager uses the physical resources that are switched on according to the decision made by the mid-term capacity manager, that, in its turn, manages the physical resources that are available as the result of the decision made by the long-term capacity manager. In this paper we focus on the short-term capacity planning.

By analyzing other short-term capacity management solutions in depth we can compare them regarding two aspects: operation mode and level of intrusiveness. Some short-term managers are proactive in their mode of operation [5], trying to anticipate future load and proactively increase or decrease services' capacities. Others only react to load changes [1], [3], and others follow a hybrid proactive-reactive approach [4] in which predictions are made, but the manager reacts as quick as possible to wrong capacity planning decisions. Our mechanism follow a hybrid approach, similar to the one proposed by Urgaonkar et al. [4].

Regarding intrusiveness, some short-term managers are service specific [4], [13], [1], in the sense that they need specific information about the services being scaled, such as response times, queue lengths, arrival rates and request demands. This requirement not only raises privacy issues, but also brings more complexity to the management framework, since the service must be instrumented to be monitored accordingly. On the other hand, there are managers that assume knowledge about the model of the service being scaled out. For instance, Calcavecchia et al. [3] define a short-term manager that only needs to monitor the load of the VMs, but requires the services to be batch jobs. We propose here a service agnostic mechanism that does not require specific service information to be monitored nor well known service models. Our mechanism only needs to know historical information about the use that the services make from the infrastructure, which can be easily collected at the VM level.

As far as we know, the auto-scaling mechanism we propose in this paper is the first to be completely independent of the service in a scale out scenario. Besides, it follows a proactive-reactive operation mode and the proactive behavior considers the use of multiple predictors, from which one is judiciously selected from time to time to provide estimation about future service demand, allowing auto-scaling to be adaptive to changes on load patterns.

III. Problem definition

We consider that the data center that supports an IaaS deployment model is comprised of a number of clusters that use some virtualization technology. Those clusters host VMs that run services of the IaaS provider users. The data center defines a set of instance types that can be instantiated and started on top of its clusters. Each instance type characterizes a VM in terms of resource capacity (CPU, RAM memory, network bandwidth, and so on).

The services that run on the cloud infrastructure are composed of one or more layers, or tiers. Each of these layers may present time varying workloads, which means that the amount of VM instances needed to appropriately run a service layer may vary over time. Thus, each layer of a service is seen as a component to be dynamically provisioned with resources. We assume that workloads are horizontally scalable, and a load balancing service is able to appropriately balance the demand of each layer on all VMs that have been allocated for that layer. For the sake of simplicity, we assume that each service layer is associated with an instance type, i.e. different service layers may be executed by different instance types, but each service layer may only run at instances of the same type. If different VM types were used it would be possible to tune the

infrastructure in a more accurate fashion, even when large instances are in use. However, it would bring a complexity not only to the scaling out service but also to the load balancer, as both would have to deal with heterogeneous resources accordingly. We believe the gain achieved by using heterogeneous resources in a layer does not compensate the extra complexity.

Each layer of a service is associated with one or more service level objectives (SLOs). We assume that there is a mapping that relates the satisfaction of the SLOs and the levels of resources utilization of the allocated VMs, in such a way that if the resources utilization is kept sufficiently often below some given targets, then the SLOs of the service layers are always satisfied. Clearly, keeping the resource utilization close to the target will result in needing fewer VMs and reduce cost.

Therefore, the goal of the auto-scaling mechanism is to perform short-term planning of each service layer of each service running into the data center, with the goal of minimizing the amount of active resources used, while meeting the layers' SLOs.

IV. Autoflex: A framework for service agnostic auto-scaling

A. Architecture

The solution that we propose, named Autoflex, is based on a standard control feedback loop. Figure 1 depicts our conceptual architecture. We consider that the system is augmented with a *monitor* component that is able to gather information about the utilization of resources at the VM level. The goal of the system monitor is to collect resource utilization information about each active VM. We assume that it is able to monitor a restricted group of resource types, such as CPU, memory, network bandwidth, operating system queues, etc. Information about different resource types can be collected for each layer of each service, with different monitoring periodicities. The utilization information gathered is periodically confronted with input reference values (e.g. target utilizations) defined by either the cloud administrator, or the user, or both. The differences between the monitored values and the reference values, known as the measured error, are fed to a *controller* component that may actuate on the system in order to bring it to a desirable state, where these errors get closer to zero. For the problem we are addressing, the controller actuates on the system by booting up new VMs or shutting down running VMs, to maintain resources utilization as close as possible to the targets associated to those resources.

The controller operates in both reactive and proactive ways. Inside the controller there are layer controllers that run periodically, in configurable intervals, to perform short-term capacity planning at the layer level. The reactive behavior is implemented by defining actions associated to each type of resource used to control a given layer. These actions are

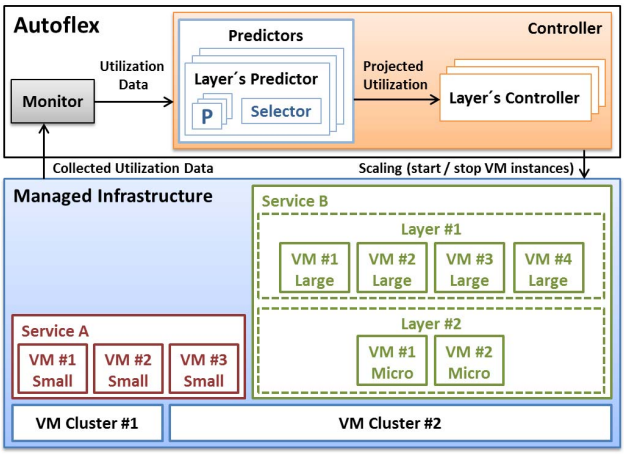

Figure 1. Autoflex architecture.

performed whenever the utilization of the resource reaches a given threshold. The proactive behavior is accomplished as follows. For each service layer, and each resource type used to control a particular layer of a service, there is a set of predictors that can be used to predict the future utilization of the resource for that layer. These predictors consume monitoring information about the resource utilization of the layer, and based on that information estimate the future load demand that the layer will place on that specific resource. These independent predictors differ only in the prediction strategy they use, i.e. how the predictor uses the monitoring information to estimate the future load. Predictors subscribe to the system monitor to receive the specific information they need. Different layer controllers may use different sets of predictors.

At each instant in time, and for each resource used to control the capacity of a layer, only one of the independent predictors is used to predict the future utilization of the corresponding resource. From time to time, a *selector* module tries to find out the most appropriate predictor to be used in the near future, from the set of available predictors used for a particular layer. The strategy of the selector module is configurable. This selection is important because, as we will show shortly, there is no single predictor that is efficient at predicting the wide variety of utilization patterns found in practice, both across layers, as well as within a single layer, when considering longer time frames.

When executed, each layer controller calculates the number of VMs needed during the next control interval for running the layer, based on the reference utilization targets, and the predicted future demand values for all the resources used to control the layer. If the number of VMs needed is larger than the number of VMs currently allocated to run a layer, the additional VMs are started. Otherwise, the control action to be performed depends on the deployment

model practiced by the provider. In a public IaaS deployment model, for instance, the provider usually charges a fixed price for any VM that is used for any part of a predefined interval of time, e.g. US$ 0.10 per hour. In this case, the controller should only shut down VMs when a new hour is to be completed. On the other hand, in a private IaaS deployment model, VMs should usually be shut down as soon as possible. Obviously, there must be some orchestration among these layer controllers in such a way that the decisions made by one controller do not affect the other layers of the same service. At a minimum, the concurrent requests for VM instance creation must be properly satisfied by the underlying physical infrastructure. Evidently, more sophisticated orchestration can be implemented in order to further improve the system's efficiency. This is, however, outside the scope of this paper.

Different from other approaches, Autoflex does not require any information about the service itself, apart from the mapping that relates a particular VM configuration with each of the layers of the service, and performs capacity planning solely based on the historical information about the use that the service layers made from the infrastructure. In this sense, Autoflex is agnostic to the service that is running in the infrastructure. Also, Autoflex provides a lot of flexibility in the way utilization predictions are made, as it can incorporate as many predictors as needed, on a per layer basis, which are dynamically selected over time.

B. A case for dynamically choosing among multiple predictors

1) Simulation model: Our decision to use multiple predictors that are dynamically selected over time is based on empirical evidences that this is a better alternative than having just a single predictor, even if the predictor embodies characteristics of several predictors, such as when an ensemble approach is used [6].

Our evaluation methodology is based on trace-driven simulations. We have implemented a simulation model of the framework presented in the previous subsection, using the R statistical environment [14].

The simulator is fed with data from production traces of HP customers. We considered that each trace corresponds to a different service layer. Since each layer controller is independent, and we are not concerned with the capacity planning of the IaaS provider, each layer controller is simulated separately.

The traces provide data that correspond to the average CPU utilization for 5-minute intervals collected at every 5 minutes. Each data item in the trace is a pair (u, c), where u is the 5-minute average CPU utilization, and c is the number of CPU cores that have been used during the last 5 minutes. A new data item is read at every 5-minute interval, and this data item is used to compute the real utilization of the VMs in the simulated system for the next 5 minutes. We consider VMs with a single CPU core. Let t be the time when a new data item (u, c) is read, and a the number of VMs that are currently allocated to run the service layer in the simulation experiments. The real utilization of each VM allocated to the layer from the instant of time t until the instant of time $t + 5$ minutes is computed as the minimum between 100% and $u \cdot c/a$. Moreover, if $u \cdot c/a > 1$, then the excess demand that could not be handled by the capacity allocated at t (given by $1 - u \cdot c/a$) is added to the demand of the next time interval (that occurs at $t + 5$ minutes).

A scaling decision is taken proactively at every 5 minutes, and reactively, whenever an action is activated, i.e. the current simulated CPU utilization is above a certain threshold. However, since it takes some time for the provisioning of new VMs to take effect, in the simulation model we consider that if the layer controller decides at time t that a new VM should be allocated, that VM will only be operational at time $t + 5$ minutes. Thus, in the case of proactive scaling, the predictors embedded in the layer controller use historical information collected up to time t to predict the layer demand at time $t + 5$ minutes, and use this information to decide at time t, how many VMs should be running at time $t + 5$ minutes. Any extra VMs required at time $t + 5$ minutes are requested at time t. We used 5-minutes intervals to be in accordance with the traces we had to evaluate the framework. However, this interval is configurable and, ideally, may be representative of the time needed to prepare new VMs to run the service layer.

In the simulation experiments reported in this paper we only considered the public IaaS deployment model, and a pricing model that charges the usage of VMs by the hour. That is to say, VMs are only shut down when a full hour has elapsed, as discussed in Section IV-A.

2) Different layers require different predictors: For this initial study, each layer controller has a single predictor, therefore, the selection mechanism is trivial. The controller strives to keep the CPU utilization below a target that was set to 70%. We have evaluated 5 popular predictors, that are based on: auto-correlation (AC), linear regression (LR), auto-regression (AR), auto-regression with integrated moving average (ARIMA), and the previous utilization measured (dubbed Last Window, or simply LW). We have also evaluated a sixth predictor that uses the other 5 predictors in an ensemble (EN), as proposed by Jiang et al. [6].

We used 265 production traces of one month duration as input to the simulator. The cumulative distribution function of the CPU utilizations of these traces can be seen in Figure 2. It can be seen that the traces represent a wide variety of utilization distributions. This is also the case for the cumulative distribution function of the CPU utilization variations, i.e. the difference between the utilization at time t and the utilization at time $t + 5$ minutes. Thus, we consider the set of traces used as representative for the study we conducted.

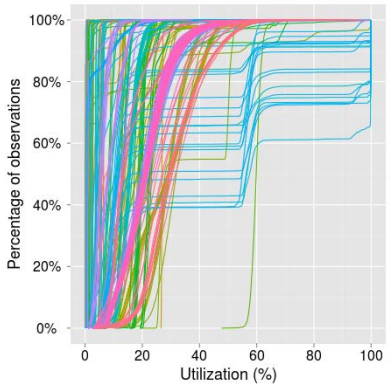

Figure 2. CDF for the CPU utilization of 265 one month duration traces.

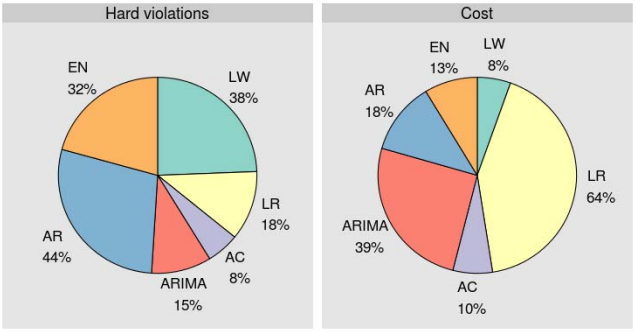

Figure 3. Percentage of traces for which a given predictor is the best predictor, considering the 265 different traces studied.

For each trace simulated, we measured the number of 5-minute intervals that the CPU utilization was at 100%, and there was still some demand that could not be handled, and spilt over the next interval. This represents the number of time intervals where the capacity allocated was not enough to handle the service demand[1]. We named these events *hard violations* of the service's SLO. We have also measured the cost reduction that is achieved when we compare the auto-scaling provisioning against that of a perfectly informed over provisioning approach, i.e. one that knows what is the highest demand within the simulation period and statically allocates resources so that the utilization is at most the required target (70% in our experiments). These metrics were independently used to rank the performance of the various predictors evaluated. In Figure 3 we present, for each predictor, the percentage of the traces for which the given predictor was evaluated as the best predictor for a trace[2], considering both metrics.

As it can be seen, when we consider the number of hard violations, the predictor that performs best responds only for 44% of the cases, while this number increases to 64%, when the cost metric is used to rank the predictors. Thus, using multiple predictors have a direct impact on the performance achieved by the auto-scaling mechanism.

3) A single layer requires different predictors over time: We have also evaluated how the quality of the predictor varies over time. For that we have fed the simulations with 33 production traces that, on average, span for an 8-month long period, approximately. The characterization of these traces also showed a wide variety of utilization partners. For the sake of conciseness, we refrain from presenting the CDF for the utilization, and variation of the utilization for these traces.

[1]We note that this is only possible because we are in a simulated environment. In a real setting, there is no way to measure the actual demand, if the utilization is at 100%.

[2]Notice that the sum of the parts in the pie chart may be larger than 100% because for some traces, several predictors may be equally good.

In this case, the set of predictors include all six predictors used in the previous experiment. A new predictor is selected whenever a 24-hour period of time has elapsed. A perfect selector is used to select the most appropriate predictor. This is achieved by feeding predictors with the past two weeks of utilization data and, using the utilization of the following day to measure the performance of each predictor[3]. The selector then chooses the best predictor, according to the metric chosen. Figures 4 and 5 present, respectively, the cumulative distribution function of the number of different predictors chosen per trace and the number of times the predictor to be used changes over time for the 33 traces simulated.

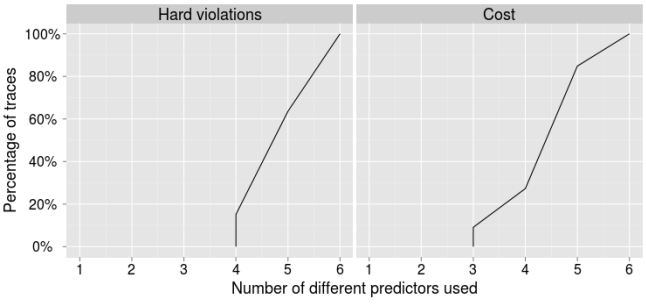

Figure 4. Number of different predictors chosen by the selector.

We can see that at least 3 different predictors are always used, and the number of times that the predictor is changed varies from dozens to hundreds of times during the 8 months period. This is an indication that the performance of an auto-scaling mechanism can be improved if predictors are dynamically selected to be used, as time advances and utilization patterns change.

[3]Note that this selector is not practical, since it requires perfect knowledge about the future utilizations; nevertheless, it is useful for the purpose of the evaluation that we perform in this experiment.

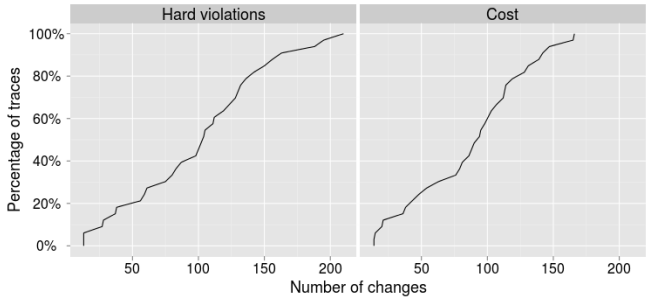

Figure 5. Number of times that the selector changes the predictor from one predictor to another in consecutive decisions.

V. Instantiation and evaluation of an Autoflex auto-scaling mechanism

In the previous section we have presented a general framework for the implementation of auto-scaling mechanisms. In this section we evaluate one possible instantiation of the proposed framework.

The instantiation of the framework requires the following actions in order to precisely define how the controller will operate: i) define threshold and actions associated with the reactive behavior of the controller; ii) provide implementations for the set of predictors to be used by each layer controller; iii) provide the implementation of a selection mechanism to dynamically choose the predictor to use at each instant in time; iv) define the appropriate values for the framework parameters, such as the periodicities with which the layer controllers and the selector are executed, as well as monitored information is gathered.

In the following we propose a way to implement a selector, predictors that try to reduce the errors that lead to under provisioning, and simple reactive actions, that together are used to instantiate an auto-scaling mechanism that aims at substantially reducing the number of hard violations, yet without compromising too much the costs savings that can be achieved through the auto-scaling approach.

A. How to choose the appropriate predictor?

Our implementation of the predictor selection module uses historical utilization data to assess which of the predictors available would perform best, based on the near past. For that, we use the last three weeks of utilization history. The oldest two weeks of data are used to feed the predictors, while the newest week of data is used as ground truth to assess the performance of the predictors. The predictor that provides the best performance in terms of the number of hard violations (and costs, for breaking ties) is selected to be used until a new selection is performed, which is done on a daily basis.

To evaluate the performance of this selector, we have run simulations using the same configurations described in Section IV-B3. In addition to the selector presented above, we considered two other selectors for comparison: the unrealistic perfect selector described in Section IV-B3, and a random selector, which randomly selects a predictor to be used each day. These can be seen as upper and lower bounds, respectively, on the performance that can be achieved by any sensible selector. In Figure 6 we show the box plot of the number of hard violations when simulating the 33 longer duration traces. As can be seen, the proposed method is slightly worse than the perfect selection, and much better than the random selection. For all cases, cost reductions compared to an over provisioned system are almost the same, around 65%.

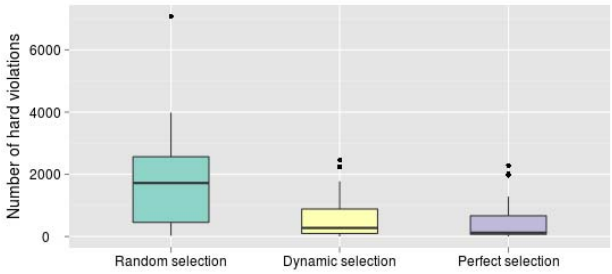

Figure 6. Evaluation of the dynamic selection method according to the number of hard violations detected in comparison with random selection and perfect selection.

B. Improving predictions to avoid underestimations

We notice that even if the perfect selector were used with the set of standard predictors that we consider in this paper, the number of hard violations may still be too large. SLO violations, characterized in our study by the number of hard violations, may lead to SLA breaches that are often expensive. Thus, in many cases, it is important to reduce as much as possible the number of such events, even if this implies in an increase on the cost of provisioning. One way to achieve this goal in proactive auto-scaling, is to use more conservative predictors. Essentially, these are predictors that try to correct their predictions, based on the previous errors made. In particular, considering those errors that led to underestimation of the capacity required, and, consequently, to hard violations [7].

Here we propose an approach to correct predictions, so to reduce the number of underestimations. The correction procedure calculates the history of prediction errors and tries to correlate the recent sequence of prediction errors with sequences of prediction errors in the past. The point in the past where the correlation is maximal is used as the center point of a window of error values. Then, the largest negative error within this window is used to correct the next prediction. Both the sequence size, as well as the window size are parameters of the mechanism, and we have used a

sequence size of 2 weeks, and values for window sizes that range from 4 hours to 2 days.

Figure 7 shows the box plot for the number of hard violations in simulations fed with the 33 long duration traces, using a setting similar to the one used in the previous experiment, but including in addition to the six original predictors, versions of all predictors with the correction method proposed above, and using different window sizes (referred to as 'ACF-x' in the figure, where x is the size of the window in number of 5-minute intervals). We also show the performance attained when the correction method used is the one proposed by Gong et al. [7] (referred to as 'Padding'). Although this method has been proposed in a different context, it is used here for comparison.

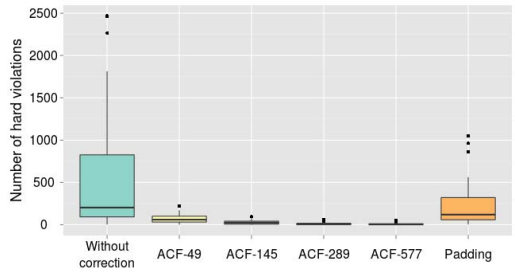

Figure 7. Comparison of two methods of prediction correction to avoid underestimation w.r.t. the number of hard violations measured.

It can be seen that all correction methods are able to substantially decrease the number of hard violations, with the 'ACF-x' method being more efficient for the setting we are addressing. Not surprisingly, considering the number of hard violations, the performance of the 'ACF-x' method improves as x increases. Figure 8 shows the corresponding cost reductions for the same experiments. As expected, reductions on the number of hard violations come at the expenses of an increased cost.

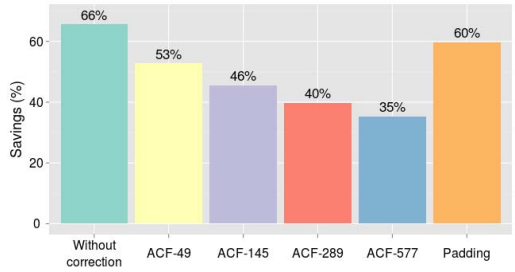

Figure 8. Comparison of two methods of prediction correction to avoid underestimation w.r.t. resource savings.

C. Putting everything together

One can see the correction of the predictions discussed in the previous subsection as a new kind of predictor. By doing that, it is possible to easily consider corrections in the Autoflex framework, by simply including in the set of available predictors, some predictors that implement correction procedures. Then, different results can be achieved, when one is more or less conservative in the configuration of the Autoflex instance.

In the following we evaluate the trade-offs involved in configuring an Autoflex instance. We have run simulation experiments that do proactive auto-scaling using the selector presented in Section V-A, and different sets of predictors. Let P be any of the predictors in the set {AC, LR, AR, ARIMA, LW, EN}, and P-x be a predictor that uses the predictor P, and performs corrections using the ACF-x method with a window of size x. We have used the following sets of predictors: C_1={P, P-49}, C_2={P, P-49, P-149}, C_3={P, P-49, P-149, P-289}, C_4={P, P-49, P-149, P-289, P-577}, C_5={P-49, P-149, P-289, P-577}, C_6={P-149, P-289, P-577}, C_7={P-289, P-577}, C_8={P-577}. In all cases, reactive auto-scaling is triggered whenever a hard violation occurs. In this case, the predictor that makes the largest corrections is used until the current 24-hour period is over, and a new selection is made by the proactive auto-scaling mechanism.

Figures 9 and 10 present the performance of different configurations with respect to the number of hard violations incurred, and the costs reduction yield, respectively. We also present the performance of the configuration that does not use error corrections, to serve as a baseline for comparison.

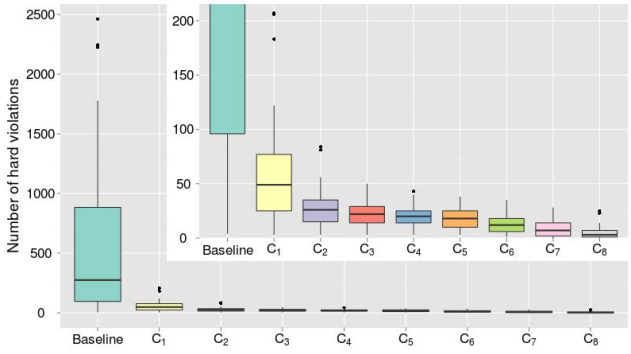

Figure 9. Comparison of different configurations w.r.t. the number of hard violations measured.

As it can be seen, the configurations that contain P-577 (C_4 to C_8) are the ones that provide the greatest reductions on the number of hard violations. These configurations produce an average number of hard violations that is between 31 (C_4) to 122 (C_8) times smaller than the baseline, with average probability of SLO violation of 0.032% and 0.008% respectively. In the worst case, configuration C_4 leads to a probability of 0.062% of SLO violation, while configuration C_8 leads to 0.036%. This also comes with a reduction on the average costs savings achieved that varies from 29% to 43%, when compared to the average cost savings achieved in

the baseline scenario, but that still yields substantial average costs savings, varying from 37% to 46%. On the other hand, the other configurations have smaller reductions on the number of hard violations, varying from a factor of 10 to a factor of 27, with reductions on the average costs savings achieved not larger than 28%.

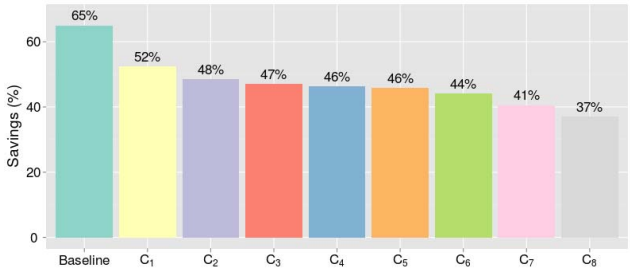

Figure 10. Comparison of different configurations w.r.t. costs reductions.

VI. Conclusion and Future Work

We have proposed a new framework for the implementation of auto-scaling mechanisms that follow both reactive and proactive approaches, and is service agnostic. The flexibility and efficiency of the framework have been demonstrated through a number of simulation experiments driven by production traces of HP customers. The results of these experiments show that a suitable instantiation of the proposed framework is able to reduce the average probability of hard violations to occur to as little as 0.008%, still providing average costs savings of 37%, when compared to an over provisioned system. In this case, the probability of hard violations to occur is always kept below 0.036%. More substantial savings can be achieved with only a minor increase on the probability of hard violations to occur.

As future work, we are evaluating how the performance of the auto-scaling mechanism can be improved by combining utilization data from several different resources. In addition, we are implementing the proposed framework to evaluate its performance in a real setting.

Acknowledgment

This research was conducted in cooperation with Hewlett-Packard Brasil Ltda. using incentives of the Brazilian Informatics Law (Law N. 8.2.48 of 1991). Francisco Brasileiro is a CNPq/Brazil researcher (grant 305858/2010-6).

References

[1] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 43–52.

[2] S. Vijayakumar, Q. Zhu, and G. Agrawal, "Dynamic resource provisioning for data streaming applications in a cloud environment," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 441–448.

[3] N. Calcavecchia, B. Caprarescu, E. Di Nitto, D. Dubois, and D. Petcu, "DEPAS: a decentralized probabilistic algorithm for auto-scaling," *Computing*, vol. 94, pp. 701–730, 2012.

[4] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, no. 1, pp. 1:1–1:39, Mar. 2008.

[5] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 559–570.

[6] Y. Jiang, C.-s. Perng, T. Li, and R. Chang, "Self-adaptive cloud capacity planning," in *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 73–80.

[7] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for cloud systems," in *Network and Service Management (CNSM), 2010 International Conference on*. IEEE, 2010, pp. 9–16.

[8] P. D. Maciel-Jr., F. V. Brasileiro, R. V. Lopes, M. Carvalho, and M. Mowbray, "Evaluating the impact of planning long-term contracts on the management of a hybrid IT infrastructure," in *Integrated Network Management*, 2011, pp. 89–96.

[9] M. Maurer, I. Brandic, and R. Sakellariou, "Enacting slas in clouds using rules," in *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 455–466.

[10] 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*. New York, NY, USA: ACM, 2011, pp. 5:1–5:14.

[11] J. Almeida, V. Almeida, D. Ardagna, I. Cunha, C. Francalanci, and M. Trubian, "Joint admission control and resource allocation in virtualized servers," *J. Parallel Distrib. Comput.*, vol. 70, no. 4, pp. 344–362, Apr. 2010.

[12] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 644–651.

[13] Y. Seung, T. Lam, L. E. Li, and T. Woo, "Cloudflex: Seamless scaling of enterprise applications into the cloud," in *INFOCOM*, 2011, pp. 211–215.

[14] "The R project for statistical computing," http://www.r-project.org/, accessed: 19/11/2012.