An Autonomic Auto-scaling Controller for Cloud Based Applications

Jorge M. Londoño-Peláez
Escuela de Ingenierías
Universidad Pontificia Bolivariana
Medellín, Colombia

Carlos A. Florez-Samur
Netsac S.A.
Medellín, Colombia

***Abstract*— One of the key promises of Cloud Computing is elasticity – applications have at their disposal a very large pool of resources from which they can allocate whatever they need. For any fair-size application the amount of resources is significant and both overprovisioning and under provisioning have a negative impact on the customer. In the first case it leads to over costs and in the second case to poor application performance with negative business repercussions as well. It is then an important problem to provision resources appropriately.**

In addition, it is well known that application workloads exhibit high variability over short time periods. This creates the necessity of having autonomic mechanisms that make resource management decisions in real time and optimizing both cost and performance. To address these problems we present and autonomic auto-scaling controller that based on the stream of measurements from the system maintains the optimal number of resources and responds efficiently to workload variations, without incurring in over costs for high churn of resources or short duration peaks in the workload.

To evaluate the performance of our system we conducted extensive evaluations based on traces of real applications deployed in the cloud. Our results show significant improvements over existing techniques.

Keywords—autonomic resource management; cloud computing

I. Introduction

Cloud Computing has radically changed the way we provision computing resources. In The Cloud one can allocate resources in a matter of seconds, use them for as long as they are needed and then release them, making it possible a pricing model where you pay for what you use. The possibility of allocating and releasing resources on demand from what seems an unlimited pool is called elasticity. Both characteristics pose interesting new problems for system administrators: How to efficiently manage resources in a cloud environment.

Ideally, the burden of dynamically managing resources in a cloud environment should be automatized so that system administrators do not have to worry for the significant workload changes that may occur in a short term basis, typically minute by minute. To do so, some service providers offer some form of auto-scaling controller that takes as input a set of measurements from the system and makes decisions about allocating or releasing resources as needed. The auto-scaling controller itself depends on a set of control parameters that determine how fast it will react to changes, how well the allocated resources will match the workload on the system, and how much will it cost to operate the whole system.

Determining an optimal set of parameters for the auto-scaling controller is not a trivial task and it commonly done by trial and error. The main contribution of this paper is posing the auto-scaling parameter determination problem as an optimization problem that can be numerically solved and used in practical systems so that the auto-scaling controller operates in a self-managed manner. We also present an implementation of the system and tests performed on real workload traces that show it benefits.

In §II we present our problem and its application scenario. §III describes the design of our auto-scaling controller and §IV explains the self-tuning technique adopted for our controller. In §V we present the evaluation of the system based on traces of actual cloud workloads. Finally, in §VI we review the related work and highlight the differences with respect to our mechanism.

II. Context

We consider the problem of dynamically allocating resources for an application deployed in the cloud when resource requirements are not known in advanced, exhibit high variability, and are difficult to predict. This context precludes the use of schedule-based allocation mechanisms. Henceforth, we concentrate in the problem of dynamically allocating resources in an Infrastructure as a Service (IaaS) cloud taking as input various performance measurements of the system.

A. *Resource allocation and measurment*

It is customary that IaaS providers handle resource allocation in the form of Virtual Machines (VMs). There are usually several VM sizes and the cloud provider has no access to the guest OS running on those VMs. Therefore the monitoring infrastructure only registers variables available at the hypervisor level. We refer to this kind of measurements as blackbox metrics. It is also possible to have monitoring agents running inside the VMs which would have access to application dependent metrics. In this case they would be called whitebox metrics.

For the purpose of our proposed system we will be using only blackbox metrics; although we hypothesize the general principles stated in this paper could be extended to handle the case of whitebox metrics as well. Validation of this hypothesis is left for future work.

B. *Cost model*

Standard per hour pricing policy is to charge a fixed amount per hour or fraction of resource usage. The hourly rate $\propto_{T(i)}$ is determined by the type of the i^{th} resource, namely T(i). Thus, the cost of n resources is the summation

$$cost = \sum_{i=1}^{n} \propto_{T(i)} \left\lceil \frac{t_{termination,i} - t_{start,i}}{1\ hr} \right\rceil \qquad (1)$$

where $t_{start,i}$ and $t_{termination,i}$ are the times at which the i^{th} resource is started and terminated, respectively. The ceil operation captures the fact that the customer is charged the hourly rate if the resource is used for a fraction of an hour. The key observation from this equation is that churn is very inefficient. For example, launching two instances for a few minutes each would cost twice as much as launching one instance for one hour. This insight will be fundamental in supporting the termination policy used by our controller, as discussed in the next section.

III. IMPROVED AUTO-SCALING TECHINIQUE

Fig. 1 shows the overall architecture of an auto-scaling system. Its main components are:

1) *Smoothing*: It is responsible of obtaining measurements from the monitoring service, applying a smoothing filter (e.g. SMA, EWMA), and computing an estimate of the workload to be used for auto-scaling decisions. Other proposals [1] include the use auto regression techniques to compute these estimates.

2) *Controller*: Given the estimated workload it determines the optimal number of instances required. The optimizer solves the problem of minimizing the total cost of the service while satisfying the application requirements.

3) *Resource allocator*: Takes the number of instances requested by the controller and instructs the management infrastructure to launch/terminate instances as required.

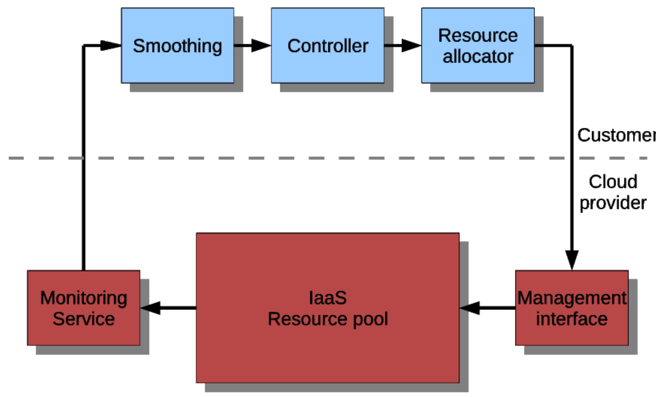


Fig. 1. Architecture of the auto-scaling system

A. *Smoothing*

Workload measurements typically exhibit high variability which makes resource management at small time-scales not feasible. Launching a new instance typically takes from tens of seconds to minutes and the workload contains many short duration spikes. Instead of making allocation decisions based on short duration spikes the controller needs to identify workload variations that will persist for long enough periods of time in order to launch or terminate VMs.

Low-pass filters such as the Simple Moving Average (SMA) or the Exponentially Weighted Moving Average (EWMA) are commonly used to smooth the inputs. The SMA filter is defined as the average of the last m samples of a metric, being m the parameter to be optimized. Large values of m represent a large smoothing effect and slow response to changes. On the other hand, for small values of m the smoothed signal will closely follow the metric and give fast response to rapid changes. The EWMA filter weights the history of the signal by a series of exponentially decreasing factors. An exponential factor close to one gives a large weight to the first samples and rapidly makes old samples negligible. On the other size, a factor closer to zero gives little weight to the last samples and makes the contribution of old samples more significant, producing a more smoothed output.

With either filter, there is a tradeoff between how fast the filter reacts to persistent changes in the input and how well it smooth out short duration spikes. Hence, the optimal filter parameter is one of the key variables to be determined using the tuning technique to be described in §IV.

B. *Controller*

It is responsible for keeping the right number of VMs in the cluster so that the resource requirements of the application are satisfied. Overprovisioning increases the cost, so the controller's job is to find the minimum number of VMs needed by the application. In addition to the number of VMs, there may be additional type restrictions, e.g. on the amount of memory, or number of cores per VM.

For the controller, we adopt a simple hysteresis controller[1] with the following parameters:

- Thresholds: Upper and lower fractions of the resource's capacity used to trigger the launch/termination of a VM.
- Update policy: How to increase/decrease the number of instances when needed. In either case, the policy can be additive or multiplicative.

Determining the optimal parameters of the controller will be part of the job of our tuning technique.

C. *Resource allocator*

The resource allocator communicates with the cloud management services to launch/terminate virtual machines as indicated by the controller. As observed in §II.B, there is no point in terminating a VM if the amount of time it has been running for is not a multiple of an hour. For this reason we adopt a *lazy termination policy* that only terminates a VM if it has been running just below a multiple number of hours and the controller is requesting a lower number of VMs. In the meantime these instances contribute to handling the workload in the cluster, thus reducing response times and providing some buffer capacity to handle short load spikes at no extra cost.

[1] essentially the same one available in Amazon Web Services (AWS)

Another feature (found for example in AWS) is the cool-down period. The cool-down period prevents the resource allocator of making any changes to the system for certain amount of time. The motivation behind is to avoid frequent creation/termination of instances when the workload exhibits high variability, as this would have a negative impact on the cost of the service. We also implemented this feature in order to evaluate its importance and the associated tradeoffs.

IV. Parameter Tuning

Clearly, the performance of the auto scaling system depends on the setting of all the adjustable parameters. In our system we have one parameter for the smoothing filter, four parameters for the controller (fixing the update policy) and one extra parameter for the cooling period. We represent the parameters as the parameter vector τ.

In order to determine the optimal setting of these parameters, we pose the following optimization problem:

$$\text{argmin}_{\theta}\{\text{cost}(\theta) + \text{penalty}(\theta)\}, \qquad (2)$$

where $\text{cost}(\theta)$ is the cost of the service as defined in (1) and $\text{penalty}(\theta)$ is a measure of the workload that exceeded the allocated capacity, i.e. it measures by how far and for how long the presented workload exceeded the allocated capacity, as given by

$$penalty = \beta \sum_{t=1}^{T} max\{0, w_t - c_t\}, \qquad (3)$$

with w_t being the workload at time t, c_t being the allocated capacity at time t, and β a weighting factor. For the purpose of the experimental evaluation we set $\propto_{T(i)} = \beta = 1$ so that a unit of excess workload costs the same as a unit of capacity. In our model time is quantized and arrival of samples from the monitoring system and the action of the controller occur with the same sampling period (5 min in our traces).

Ideally, for tuning the controller all we need to do is to solve the optimization problem (2) for the optimal value of τ over all possible inputs to the system. Unfortunately, there is no closed form description of the input and it is easy to show that the optimal solution is not unique. Instead we adopted the following technique: Given a set of traces, we split them into a training set and a testing set.

The training set is used to conduct a numerical optimization in order to find the parameter vector θ, and the test set is used for evaluating its performance. The numerical optimization uses simulated annealing, and starting from some initial parameter vector θ_0 uses a temperature parameter to obtain neighbor parameter vectors.

The minimum among the neighbors is chosen as the starting point for the next iteration. At the end of each iteration, the algorithm reduces the temperature, thus reducing the range for choosing random neighbors. After a given number of iterations of not finding improved parameter vectors, the optimization finishes.

V. Experimental Evaluation

For the evaluation of the system we used a set of traces taken from actual deployments of EC2 instances in AWS. The traces cover a period of about 15 days and include all the metrics captured by the CloudWatch service, among others, CPU utilization, disk I/O, and network utilization. In these traces disk I/O activity was minimal, thus we made no further use of these data. From these traces we took a 50hr interval as training interval, and used other intervals as test cases. As the workload in the trace is relatively small, we scaled it up by a constant factor in order to simulate larger environments.

We implemented in Matlab the standard auto-scaling system offered by AWS and our improved system. The first one will give a baseline for comparison purposes. We then used our controller tuning technique (see §IV) to obtain the optimal parameter vector for both systems with the same training trace. Then, running the test trace in both systems we obtained the performance metrics for our analysis. The standard auto-scaling controller is setup to use a SMA smoothing filter and multiplicative increase/decrease policies. Our controller uses an EWMA filter and multiplicative increase/decrease policies. Both systems implement the cool-down policy.

A. *Number of active instances over time*

A first experiment evaluates the number of active instances over time and their liveness period, i.e. the amount of time they existed in the cloud. Fig. 2 shows the number of compute units (CU)[2] for an illustrative test case. It shows both cases, standard and improved auto-scaling.

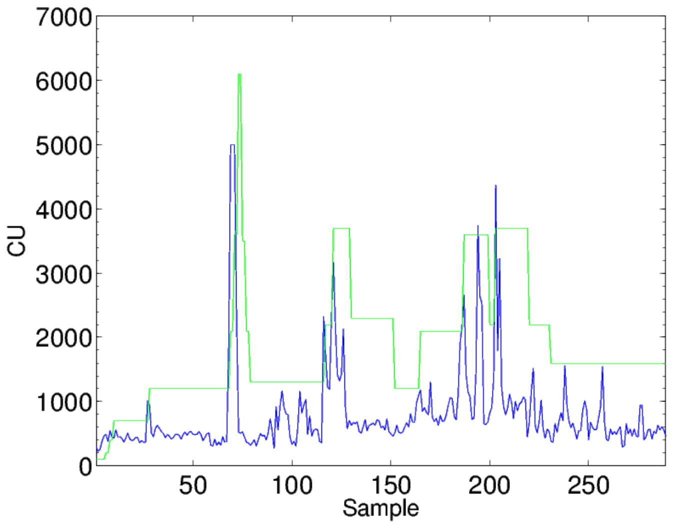
Although both track the workload (plus a safety margin as defined by the upper threshold), it is noticeable that the improved auto-scaling tracks more closely the peaks and the valleys and reacts faster to changes. This is especially remarkable in the valleys where the lazy termination policy keeps instances alive for up to a multiple of one hour, but still does a better job tracking the valleys than the cool-down policy.

The effect of the lazy termination policy is shown in Fig. 3 which shows the histogram of the liveness period of instances for the same test cases. For standard auto-scaling they take arbitrary lengths, and for improved auto-scaling they always take a multiple of twelve sampling periods minus one (12n−1). The minus one is due to our implementation terminating instances one sampling period before the hour to avoid using any fraction of the next hour.

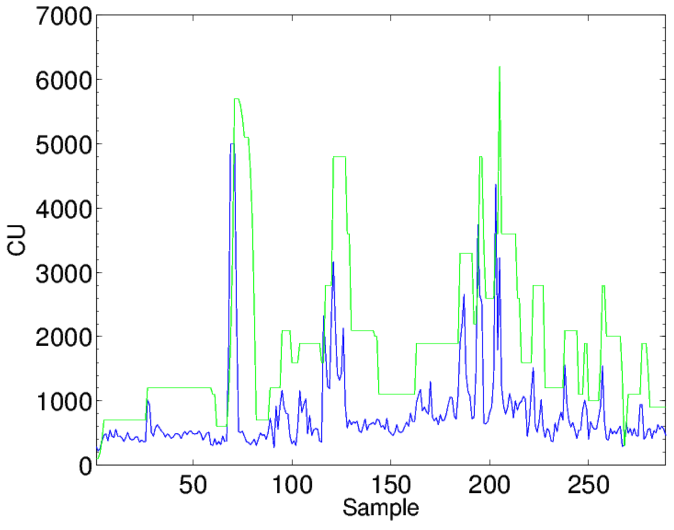
It is important to notice that our parameter tuning algorithm set the cool-down period to 3 sampling periods in the case of the standard auto-scaling and to 0 in the case of the improved auto-scaling technique. This is the result of solving the optimization (2), which in the first case is forced to keep the cool-down period larger than zero to void the problem of frequent termination/creation, which would significantly increase the cost.

In the second case, the cool-down period does not play any role in determining the cost of the service, as termination of instances is governed by the lazy termination policy.

[2] For our analysis we set 100 CU equal to 1 Amazon's ECU

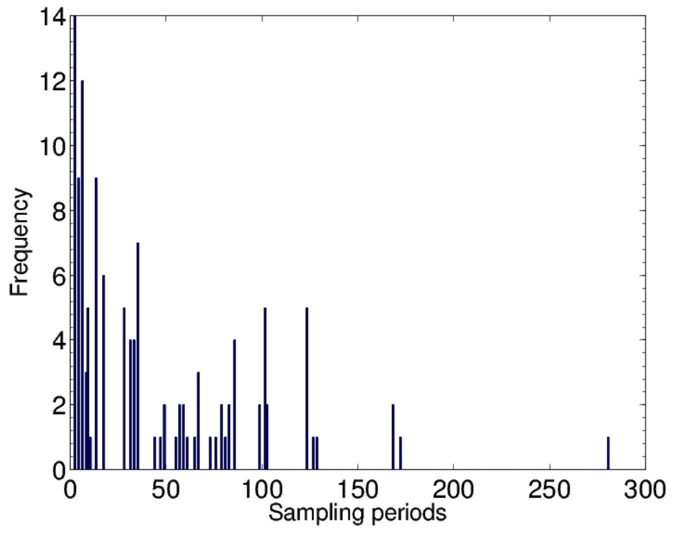


a) Standard auto-scaling

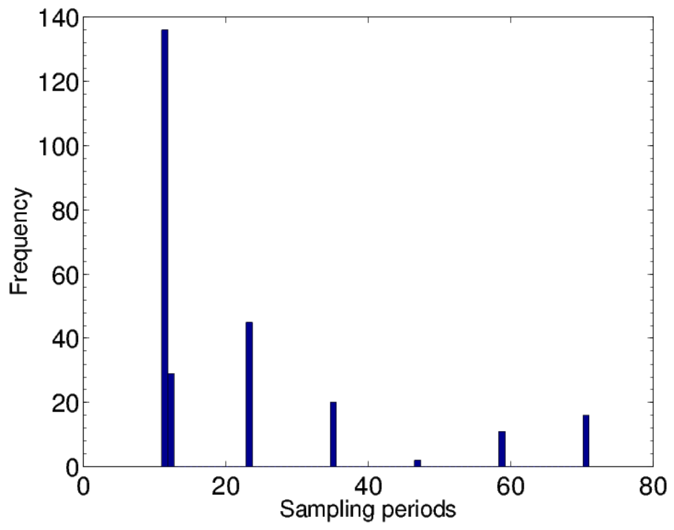


b) Improved auto-scaling

Fig. 2. Number of active instances over time



a) Standard auto-scaling



a) Improved auto-scaling

Fig. 3. Histogram of instance liveness period

B. *Cost and penalty improvements*

Fig. 4 shows the costs and penalties for all the test cases considered. Overall the improved technique has a small reduction of service cost, albeit a few exceptions. However, the comparison of penalties shows a significant reduction of the penalty with the improved algorithm. Considering all test cases, the average reduction of the cost was 6.3% and the average reduction of the penalty was 55.5%.

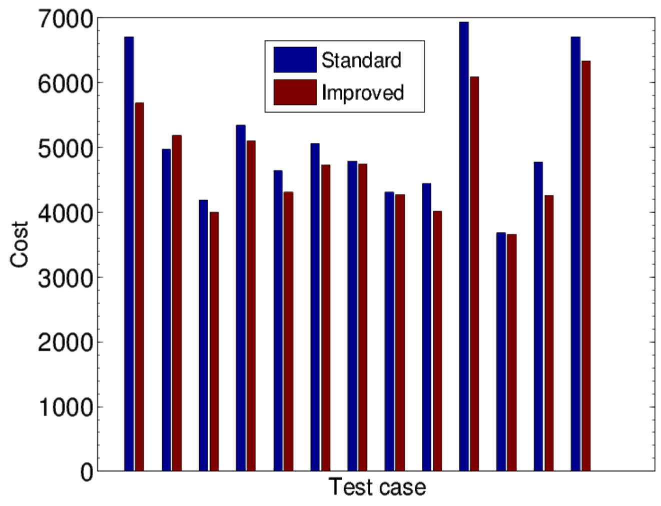
VI. Related Work

Existing literature considers various approaches for handling the allocation of resources in a cloud computing environment. Although some approaches have some features in common with our solution, there are also important differences. Following we present a brief description of the most relevant approaches and highlight the main differences with respect to our work.

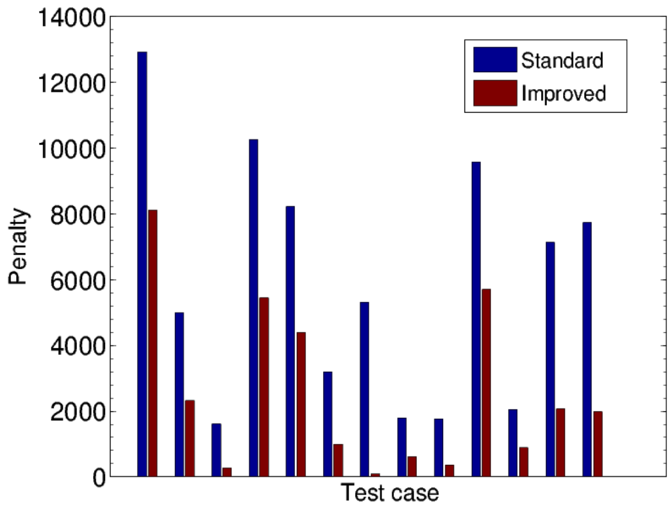
Bodík et al [2] present a technique that uses statistical machine learning to fit a non-linear performance model on the most recent set of samples. This model produces a target number of servers to satisfy the existing Service Level Agreements (SLAs), and this value is filtered through a hysteresis filter to avoid oscillations in the controller. The model captures the relationship between the number of request that fail the SLA's threshold and the current number of servers and workload. This technique does not take into account the cost of the service and under high variability on the workload would lead to frequent creation and termination of VMs, the churn problem that our technique avoids while minimizing the operational cost of the service.

Bi et al [3] developed a technique that uses a hybrid queueing model as the basis to provision resources for multi-tier applications running in a cloud data center. Their technique takes as inputs the request arrival rate, the service rate of the VMs for each tier, and the response time from the application.

Feeding this information into the model, makes it possible to determine the number of VMs required. The main difference with our system is that this technique relies on application level measurements, which may not always be available or could require and additional development and integration effort.



a) Comparison of costs



b) Comparison of penalties

Fig. 4. Comparison of costs and penalties

Padala et al [4] adopt a blackbox approach that uses control theory to manage the virtual resources assigned to the application. The controller in this system assigns entitlements of the physical resources to the VMs they host, in such a way that the application performance meets the preset SLAs. The main difference with our work is the assumption that entitlements to physical resources can be adjusted online. Although this functionality is available in several virtualization frameworks, it is not commonly offered by public IaaS providers, which prefer to offer a set of predefined instance sizes.

Turner et al [5] explore a system that builds an empirical model of the application performance. Their system is tailored for multi-tier applications running on a virtualized infrastructure. Data collected by the monitoring system includes resource consumption and application response time. A regression algorithm fits a model to the collected data and this model is used to adjust allocation of resources to the different virtual machines. Sangpetch et al [6] further develop a close-loop controller system that uses the model and a target Service Level Objective (SLO) to adjust the allocation of resources to each VM in the system. The controller uses both, a long term and a short term prediction to adjust the resource allocation to each of the VMs. These systems also rely on the assumption that the customer has control over the amount of resources assigned to each VM, which is not usually the case with IaaS providers.

Chandra et al [1] present a technique that combines measurements, a generalized processor sharing model, and time-series analysis to determine the fraction of the resources to assign to each of the application components. The allocated resources assure that the application meets its Quality of Service (QoS) constraints. This technique applies to the case of VMs sharing a host in which the entitlement of resources for each component is adjustable and the controller has access to internal performance metrics of the application. However, it is different to the problem we are handling because we deal with predetermined instance sizes, our goal is minimizing the service cost and penalty, and we restrict to blackbox metrics.

It is also worth noticing that there has been work on resource management mechanisms based on the idea of migrating VMs, as for example [7]. In this study migration was not considered as our focus is a public IaaS cloud, where migration services are not commonly available.

The problem of determining the optimal set of resources of various types with multiplicity been shown to be NP-Complete by Chang et al [8]. They also present an approximation algorithm. This algorithm considers the problem in a static setting, thus it is not applicable in the dynamic environment we consider. A related work by Dougherty et al [9] considers the auto-scaling problem from the point of view of minimizing the cost and energy consumption. In their work they used a Model Driven Engineering (MDE) approach combined with a constraint satisfaction technique to find the set of instances that supply the application requirements while minimizing cost and energy. This work assumes a static context where the application requirements remain stable over time.

VII. CONCLUSIONS

We have presented an autonomic auto-scaling controller specifically designed for allocating resources in a cloud datacenter under dynamic workloads. Our controller reduces the service cost and the performance penalties when compared to the optimized standard hysteresis controller commonly available from public cloud providers. Both characteristics are highly desirable for whoever deploys an application in an IaaS cloud.

Our controller departs from well-known auto-scaling controllers by incorporating a fast response smoothing filter, a numerical optimization technique for finely tuning the controller parameters, and implementing the lazy termination policy, which postpones the decision to terminate an instance until the very last moment possible without extra charges. Also, our experiments showed that the lazy termination policy effectively makes the cool-down period unnecessary. The cool-down period limits the response time in the event of large workload changes, thus increasing performance penalties. On the other hand, the cool-down period does not reduce the service cost when using our improved controller

REFERENCES

[1] Abhishek Chandra, Weibo Gong, and Prashant Shenoy, "Dynamic resource allocation for shared data centers using online measurements," SIGMETRICS Perform. Eval. Rev., vol. 31, pp. 300-301, 2003.

[2] Peter Bodík et al., "Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters," in HotCloud, 2009.

[3] Jing Bi, Zhiliang Zhu, Ruixiong Tian, and Qingbo Wang, "Dynamic Provisioning Modeling for Virtualized Multi-tier Applications in Cloud Data Center," in IEEE International Conference on Cloud Computing, 2010.

[4] Pradeep Padala et al., "Adaptive control of virtualized resources in utility computing environments," SIGOPS Oper. Syst. Rev., vol. 41, pp. 289-302, 2007.

[5] Andrew Turner, Akkarit Sangpetch, and Hyong S. Kim, "Empirical Virtual Machine Models for Performance Guarantees," in Large Installation System Administration Conference (LISA), 2010.

[6] Akkarit Sangpetch, Andrew Turner, and Hyong Kim, "How to Tame Your VMs: An Automated Control System for Virtualized Services," in Large Installation System Administration Conference (LISA), 2010.

[7] Mauro Andreolini, Sara Casolari, Michele Colajanni, and Michele Messori, "Dynamic load management of virtual machines in a cloud architectures," in IEEE 2009 International Conference on Cloud Computing, Los Angeles, CA, 2009.

[8] F. Chang, J. Ren, and R. Viswanathan, "Optimal Resource Allocation in Clouds," in IEEE International Conference on Cloud Computing, 2010.

[9] Brian Dougherty, Jules White, and Douglas C. Schmidt, "Model-driven auto-scaling of green cloud computing infrastructure," Future Gener. Comput. Syst., vol. 28, no. 2, pp. 371-378, 2012.