An RLS Memory-based Mechanism for the Automatic Adaptation of VMs on Cloud Environments

Carlos Ruiz, Hector A. Duran-Limon, Nikos Parlavantzas

An RLS Memory-based Mechanism for the Automatic Adaptation of VMs on Cloud Environments

Carlos Ruiz
University of Guadalajara
Periferico Nte 799
Zapopan, Jalisco, Mexico
ruiz_carlos@cucea.udg.mx

Hector A. Duran-Limon
University of Guadalajara
Periferico Nte 799
Zapopan, Jalisco, Mexico
hduran@cucea.udg.mx

Nikos Parlavantzas
IRISA INSA de Rennes
263 Avenue du Général Leclerc
Rennes, France
Nikos.Parlavantzas@irisa.fr

ABSTRACT

One key factor for Cloud computing success is the resource flexibility it provides. Because of this characteristic, academia and industry have focused their efforts on making efficient use of cloud computational resources without having to sacrifice performance. One way to achieve this purpose is through the automatic adaptation of the computational capabilities of VMs according to their resource utilization and performance. In this paper we present the design and preliminary results of our resource adaptation solution, which proactively adapts VMs (memory-based vertical scaling) to maintain an expected performance. Our solution targets multi-tier applications deployed on Cloud environments, and its core resides in RLS-based resource and performance predictors. Our results show that our solution, when compared with VMs with larger and permanently allocated computational resources, is able to maintain expected performance while reducing resource waste.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Computing methodologies** → *Machine learning approaches*; • **Computer systems organization** → *n-tier architectures*;

KEYWORDS

Cloud computing, Vertical scaling, Dynamic adaptation, Performance prediction

1 INTRODUCTION

Cloud computing is a computing paradigm whose main objective is to provide computing resources (e.g. networks, servers, storage, applications and services) on demand at low costs and in a seemingly unlimited amount. This is especially true at the Infrastructure as a Service (IaaS) level offered by the cloud paradigm, which focuses on provisioning computational resources such as processing power, network capacity and/or storage. This characteristic makes the IaaS model suitable for distributed applications (e.g., multi-tier applications), which require resource flexibility to achieve performance and cost goals, among others. Hence, academia and industry have focused their efforts on making efficient use of cloud computational resources without having to sacrifice performance or increase costs. One way to achieve this purpose is through the automatic adaptation of VM computational capabilities according to resource utilization and performance, i.e., VM resource adaptation as a response to changes in the VM environment and Service Level Objectives (SLOs). However in order to achieve such a purpose, several aspects have to be considered. Among the aspects to consider, we can mention:

- The impact on the cloud application's performance as a consequence of a modification on the supporting VM; i.e., perform adaptations on VMs without affecting agreed SLOs.
- To perform adaptation actions on a VM only when required and at the proper time. This to avoid resource waste or starvation while being compliant with agreed SLOs.

In order to face such challenges, researchers have come up with several solutions, which can be classified as follows: Regarding the time at which a change in the VM environment is detected and a consequent adaptation action takes place, solutions can be classified into reactive and proactive. *Reactive approaches*: their intention is to perform adaptation actions after a change in the VM environment has been detected. These types of approaches define thresholds on VM's allocated resources or VM's performance; once such a threshold is reached or exceeded a consequent action is taken. *Proactive approaches*: the intention is to execute adaptation actions before a resource or performance degradation on the VM affects the application running on top of it. The accuracy of this type of approaches greatly depends on the prediction technique being used and its tuning.

Despite reactive approaches being easier to implement than proactive ones, they are prone to cause SLOs violations, given that adaptation actions are taken later in time. On the other side, proactive approaches can cause resource waste or performance degradation if the prediction is not accurate. Regarding the type of adaptation action taken, the solutions use vertical, horizontal or hybrid resource escalation.

- *Vertical scaling*: it involves the increment of allocated computational resources (usually vCPU and/or memory) within a single VM at runtime, without disrupting the execution of the VM.
- *Horizontal scaling*: it is commonly used in commercial or open solutions such as AWS and Openstack [9] [5]. Its purpose is to create/instantiate new VMs, so that the currently experienced workload can be distributed among several VMs.
- *Hybrid scaling*: lately there have been efforts towards combining the previously mentioned escalation approaches. These types of solutions usually perform vertical scaling first, up to the point that it is no longer cost or performance efficient, and then horizontal scaling. The process is iterative and is repeated as much as needed.

, ,
. ACM ISBN .
https://doi.org/

Lately, hybrid approaches have received more attention from the research community, given that they try to mitigate the disadvantages that simple escalation models can have. For example, a purely vertical approach could be limited by supporting technologies e.g. not all hypervisors support transparent addition of vCPUs at runtime. However, the first step for a hybrid approach to be effective is the correct setup of its fundamental components. To that end, the current paper focuses on setting up and applying vertical scaling, which has not been sufficiently investigated in research literature. The proposed solution in this paper follows the proactive model (RLS-based prediction) in combination with vertical scaling (memory-based). Those models were chosen as fundamental building blocks given they can help to reduce resource waste and avoid performance violations. Details about these design decisions are given in the corresponding sections (see section 2). Hence, we propose a solution that targets multi-tier applications deployed on Cloud environments, which relies on RLS-based resource and performance predictors. Our preliminary results show that our solution, when compared with VMs with larger and permanently-allocated computational resources, is able to maintain expected performance while reducing resource waste. In summary, the main contributions of this work are:

- Usage of prediction techniques to vertically scale VMs.
- Coordinated usage of resource and performance RLS-based predictors to avoid resource waste while maintaining predefined performance.
- Systematic evaluation of the proposed solution.

The remainder of this paper is organized as follows. Section 2 presents the reasons for choosing RLS filters as the prediction technique and memory utilization and response time as reference indicators for predictions. The architectural design of our approach is presented in Section 3. Section 4 presents the experimental evaluation and analyses obtained results. Section 5 presents related work. Finally, Section 6 presents conclusions and future work.

2 RLS-BASED PREDICTION

This section justifies the selection of the Recursive Least Square (RLS) filter as the prediction technique and of VM memory utilization and response time as reference indicators for our RLS-based predictors (see section 3).

2.1 Recursive Least Square filter (RLS filter)

Recursive Least Square (RLS) filter is a machine learning technique that has been used for diverse purposes, such as adaptive filtering, prediction, and system identification [6] [7]. Its main objective is to minimize the sum of the square of the differences between the desired signal and the estimated filter output. When compared with Least Mean Square (LMS) filters, RLS filters offer faster convergence and smaller error with respect to the desired signal (see figure 1). A key difference between LMS and RLS filters is the fact that the latter applies a forgetting factor, which determines the influence of past input data over the current estimation. Because of those qualities, RLS-based mechanisms have been applied to the problem of dynamically adapting VM resources on cloud environments (see section 5).

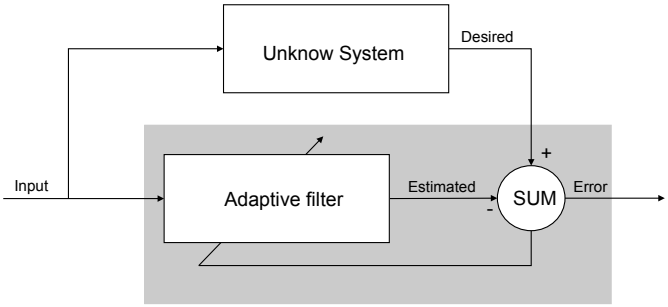

Figure 1: RLS filter

Table 1: VM tested configurations

Configuration ID	CPU (No. Cores)	Memory	HDD
MEM_C1	1	512MB	15GB
MEM_C2	1	1GB	15GB
MEM_C3	1	1.5GB	15GB
MEM_C4	1	2GB	15GB

Another prediction technique considered was artificial neural networks (ANN). Despite the fact that ANNs have been also used to address similar problems [2] [4], the final decision was taken considering the following factors:

- RLS filters dynamically adapt to the input data and produce an output accordingly, i.e., RLS are suitable for online training. On the other side, ANN prediction models are generated offline, which make them depend greatly on their training data set. Such dependency implies that training datasets have to be representative enough and to accurately describe the behaviour of the targeted system.
- RLS filters provide a forgetting factor, which can be used to determine the influence of past data over current estimations. This makes RLS filters suitable for fast adaptation to sudden changes in the input data. Conversely, ANNs have to be retrained every time the input data diverges from the data set used to train them, situation that is not desirable in highly dynamic applications such as multi-tier applications running on clouds.

2.2 Memory as prediction indicator

In order to determine which VM resource (CPU or memory utilization) better represents the behaviour of a VM under different workload conditions and, hence, can be used as reference indicator for our RLS-based predictor, we perform a profiling of a web server running on top of a VM. For the profiling process, we considered the following factors:

- VMs with different resource configurations (see table 1).
- Web server serving only static content, i.e., request response does not need to fetch data from a DB server.
- Several injection request rates (see table 2).

The profiling process on the base case configuration showed a possible correlation between VM's memory utilization and its

Table 2: Workload injection rates

Injection ID	Min req. rate	Max req. rate	Step req. rate
I1	2840 req/sec	3000 req/sec	2 req/sec

Injection ID	Req type	Max amount	Max timeout
I1	Static	25000 req	3sec

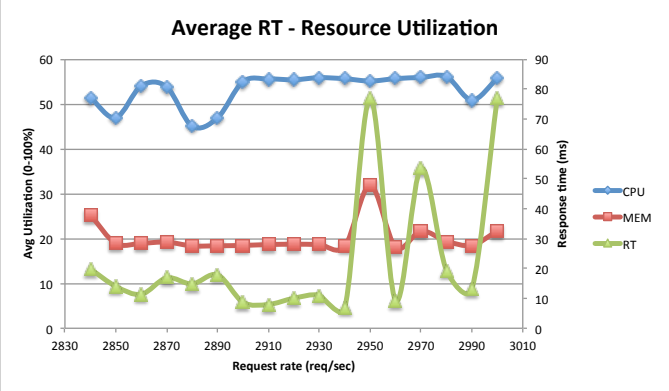

Figure 2: MEM_C3 RT - Resource utilization

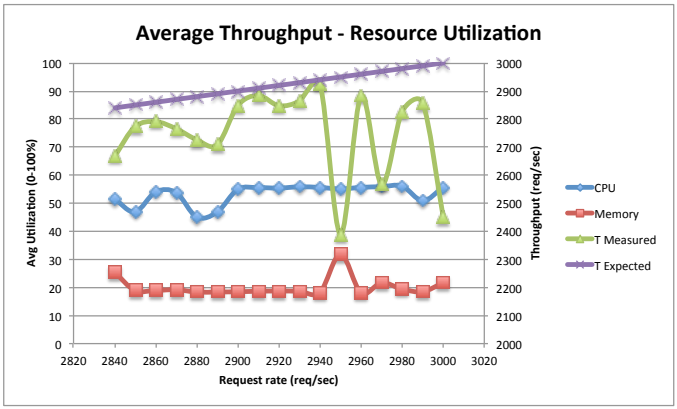

Figure 3: MEM_C3 Throughput - Resource utilization

performance. However, such correlation was not clear; therefore, we decided to repeat the profiling process using other VM resource configurations. From the results (see figures 2 and 3) we could observe the following:

- A possible direct correlation between memory utilization and the performance of the VM (in terms of response time) exists.
- A possible inverse correlation between memory utilization and the performance of the VM (in terms of throughput) exists.
- CPU utilization neither reflects the VM's experienced workload nor its current performance.

Table 3: RT - Memory correlation results

Injection rate (req/sec)	2840	2850	2860	2870
Correlation value	**0.2525**	**0.4989**	**0.4239**	**0.4675**
Injection rate (req/sec)	2880	2890	2900	2910
Correlation value	**0.6595**	**0.7093**	**0.6059**	**0.6799**

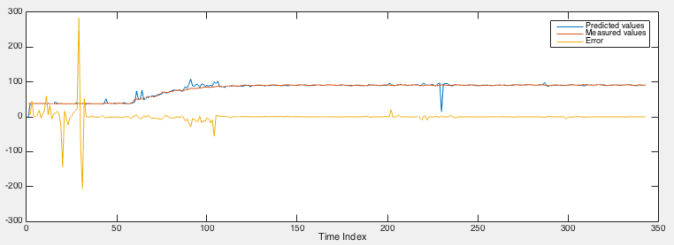

Figure 4: Prediction interval 5 seconds

In order to corroborate this possible correlation between VM's memory utilization and its performance (response time or throughput), correlation tests were applied to these variables (linear correlation test applied was Pearson's product-moment correlation with 95% confidence). However, correlation tests were not concluding given they did not show a clear correlation between both variables; we considered correlation values in the following way: values above 0.8 represent a strong correlation whereas below 0.5 describe a weak correlation. Table 3 shows correlation values for MEM_C1 (Base case) configuration. It is worth to mention that only linear correlation tests were applied. These correlation results can be explained by the fact that linear statistical methods cannot accurately describe the behaviour of multi-tier applications [1]. This explanation reinforces our decision of using RLS filters (machine learning technique) to perform our predictions.

Even though the correlation between memory utilization and performance could not be confirmed, we decided to use memory utilization for our prediction given that it showed stronger correlation than cpu utilization.

2.3 Response time as predictor indicator

The response time was chosen as reference indicator for our predictor because it is the most widely-used performance indicator for multi-tier applications in industry and academia [9] [5].

2.4 Prediction interval and accuracy of the RLS memory-based predictor

The data obtained during the profiling phase was also used to test the prediction capabilities of our RLS-memory based predictor. With this set of experiments, we were able to obtain estimations about the predictor accuracy and how far ahead it could perform predictions before producing no longer reliable results. Figures 4, 5 and 6 depict the accuracy results obtained by our RLS-memory predictor for different prediction intervals. We considered 3 prediction intervals: 5, 20 and 60 seconds.

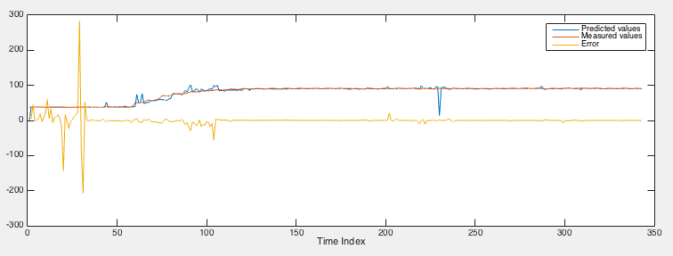

Figure 5: Prediction interval 20 seconds

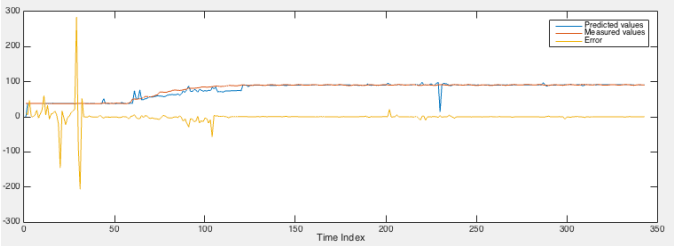

Figure 6: Prediction interval 60 seconds

Although the considered prediction intervals were similar in terms of predicted values and prediction error, the prediction interval chosen for our RLS-memory predictor was 5 seconds. The reason of such short prediction interval is that once the VM starts experimenting significant workload, its memory utilization also quickly increases (from approx. 40% to approx. 90% in less than one minute in our experiments). Hence, longer prediction intervals could lead to late adaptation actions and, therefore, SLOs violations.

Table 4 presents an excerpt of the predicted values and prediction errors obtained for the prediction interval of 5 seconds. Samples presented were taken from the time period between 50 and 120 of the time index (see figure 4). This period was chosen since it is the period of time when the VM starts experiencing significant load and, therefore, an increment in its memory utilization. Error values in table 4 give an idea about the accuracy of the RLS memory-based predictor and how fast is its adaptation process when new values are introduced as part of its input data.

3 MECHANISM DESIGN

This section describes the architectural design of our proactive adaptation solution. Figure 7 presents the mechanism's modules as well as the mechanism's flow to adapt VMs.

Our design can be divided into three main modules: Monitoring, Prediction and Execution. The monitoring module consists of two monitoring agents: Response time (RT) monitor agent and Resource monitor agent. These two agents are in charge of providing VM performance and resource utilization data to respective RLS-predictors. The resource monitor agent provides information about the current VM's resource utilization (memory utilization), and it is based on Zabbix[1] agent. The RT monitor agent is in charge of fetching information from Apache web server. Data being collected

[1]https://www.zabbix.org

Table 4: Predicted values and error (5 sec prediction interval)

Measured value	Predicted value	Error
39.229	39.3085	-0.079487
38.6283	41.2072	-2.579
38.4216	39.1293	-0.70772
39.2605	39.8073	-0.5468
41.582	42.0941	-0.51214
50.593	74.3475	-23.7545
50.3795	50.3603	0.019184
50.8077	76.6792	-25.8715
51.088	48.3209	2.7671
56.4635	56.1658	0.76153
58.799	62.2383	-3.4393

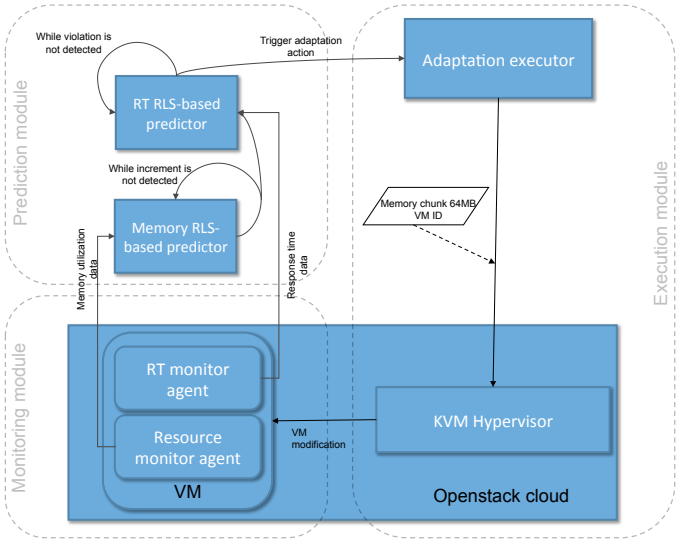

Figure 7: Architectural design

is used to derive the response time of the web server. Collected data comprehends request arrival time and request processing time. The monitor agents establish a connection with corresponding RLS-based predictors just after the VM finishes its initialization. The connection between monitors and predictors is permanent.

The prediction module consists of two RLS-based predictors, one for RT and another for memory utilization. The main objective of the memory RLS-based predictor (Memory predictor) is to determine whether an increment in the memory utilization is going to occur, event that could indicate that an increment in the RT could also occur. It starts receiving data from its resource monitor agent; received data is used to start generating an initial prediction about the behaviour of the VM's resource utilization. The period to gather this data corresponds to the *reference period* (see section 4) and lasts for approximately 40 seconds. We notice that this interval of time is enough for the predictor to generate accurate estimations. This is because RLS-based predictors use past estimated values and real values to dynamically adjust their predictions.

The RT RLS-based predictor (RT predictor) has as main objective the generation of predictions about the VM's response time. Estimations generated by this predictor are not immediately taken into consideration; they are actively considered only after an increment in the memory utilization has been predicted. However, estimations generated before a memory utilization event occurs are used as samples to increase the accuracy of the predictor. Once a memory increment has been predicted and RT predictions are being generated, we consider an *adaptation period* (see section 4) before triggering and executing an adaptation action on the VM. Regarding the execution module, as its name implies, it is in charge of performing the adaptation action on the VM. The RT predictor calls the Adaptation executor once an adaptation action has been determined. The process to execute an adaptation action is as follows:

- determine in which physical host is the targeted VM running,
- determine if it is possible to allocate more memory to the VM, i.e., verify if the maximum allowed memory has not been reached,
- instruct the host hypervisor to modify the amount of allocated memory on the VM (1 memory chunk per adaptation action).

The process of automatically adapting the VM runs as long as necessary. In case the VM's maximum allowed memory has been reached and predictions still imply a decrement on VM's performance, distinct adaptation actions have to be executed such as horizontal scaling (i.e. increasing the number of VMs), which is out of the scope of this paper.

4 EXPERIMENTAL EVALUATION

In order to verify whether our proposed solution is able to automatically adapt VMs to keep an expected performance and low resource utilization, we tested it under different workload conditions and compared it against VMs with larger and permanently-allocated computational resources.

4.1 Experiment setup

All experiments were executed on a private cloud managed by Openstack[2] Liberty running under Linux CentOS 7, which uses KVM as hypervisor. It consists of 1 controller (4 cores, 6 GB RAM, 3TB HDD) and 8 compute nodes (each compute node having 8 cores 2.13Ghz, 16GB RAM and 250GB HDD) connected through a local network 100Mbps. We chose the well-known RUBBiS benchmark as our multi-tier application. RUBBiS was deployed on top of Apache web server; for our experiments we are using only static web content. The VM used to deploy RUBBiS benchmark was initially configured with 1vCPU, 512MB RAM and 15GB HDD. Traffic generator used for our experiments was Autobench, a traffic generator tool that internally uses Httperf to generate requests and automatically tests multiple injection rates. Our solution was tested with different injection rates (see table 2), each of which was executed 25 times. The number of executions was chosen to give statistical validity to our experiments. Our results show the average values obtained for each one of the tested injection rates. In the same way as our solution was tested, we tested the other VM configurations (see table 1). This was done in order to observe the behaviour of our solution against computationally larger VM configurations and determine if it really avoids resource waste and SLOs violations. Tables 5 and 6 show used configuration parameters for our RLS-based predictors and adaptation mechanism respectively. Parameter determination was experimentally done.

[2]https://www.openstack.org

Table 5: RLS-based predictor parameters

Parameter	Value
RLS Forgetting factor	0.45
Sampling interval	Every 4 seconds

Table 6: Adaptation parameters

Parameter	Value
Reference period	20 - 40 seconds
Adaptation period	10- 20 seconds
Stabilization threshold (SBT)	Determined at runtime
Variation percentage (VP)	5% over SBT
Max allowed VM memory	2 GB
Initially VM allocated memory	512MB
Memory chunk size	64MB
VM RT sampling	Every 4 seconds
Runtime (RT) threshold	100 milliseconds

The parameters from table 6 (Adaptation parameters) are:

Reference period. Period of time used in the determination of the stabilization threshold (see below). Defined at the beginning of the monitoring phase.

Adaptation period. Period used to determine whether a variation on the resource/performance indicator is transient or not. If the variation is not transient, the adaptation process is started.

Stabilization threshold (SBT). Values used as reference points when comparing estimated values. In the case of the resource predictor, it is the Resource (memory) utilization minimum threshold. For the performance predictor, this value is equal to RT threshold.

Variation percentage (VP). Percentage used to determine whether a resource utilization increment could be considered as significant. E.g. SUT = 44, VP = 5%. Resource utilization increment that could be considered as significant = SUT + (SUT*VP), i.e. 46.2. This value is only used in the case of the memory predictor.

Total VM memory. Total memory available to the VM.

Initially VM allocated memory. Initially allocated memory to the VM.

Memory chunk size. Amount of memory to be added or removed to/from the VM in case an adaptation action is performed.

Runtime (RT) threshold. Maximum response time allowed. This value is set by the application user and is considered as a service level objective (performance measure).

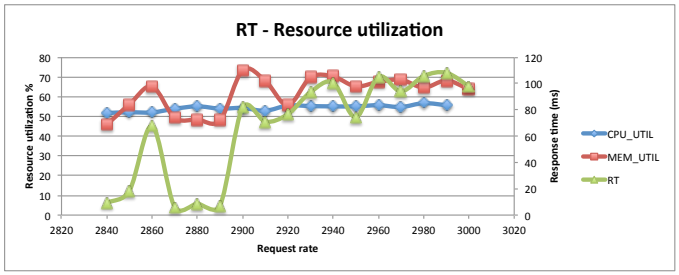

Figure 8: Response times for different injection rates

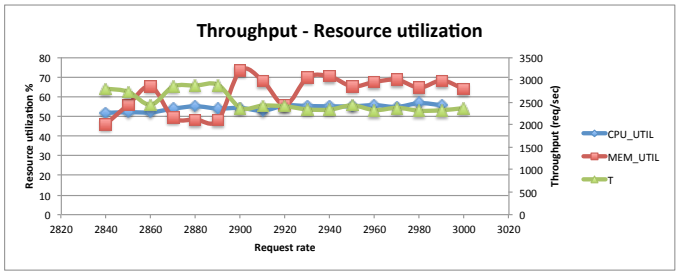

Figure 9: Throughput for different injection rates

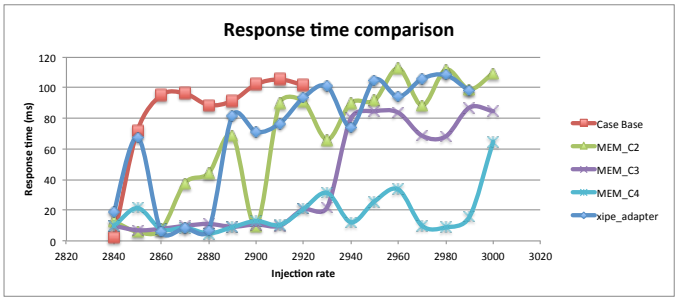

Figure 10: Response time comparison

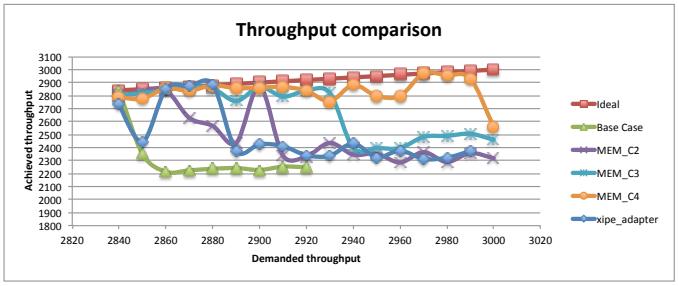

Figure 11: Throughput comparison

4.2 Results

Here we present and analyse the results obtained with our solution under different workload conditions. Figures 8 and 9 present the performance (response times and throughput) obtained with our solution in each case tested.

Results show that our solution is able to keep response times below the targeted threshold (100ms). In the case where the threshold was surpassed, it was not surpassed for more than 10ms. Also it can be observed that the memory being used by the VM was kept low, which shows that the combination of RLS-based resource and performance predictors allocates more memory only when needed. Regarding the comparison with other VM configurations (figures 10 and 11), it can be observed that our solution (xipe-adapter) outperforms the base case configuration (MEM_C1) in terms of response time and throughput. When compared with larger VM configurations (MEM_C2, MEM_C3 and MEM_C4), we can observe that its performance is similar to that of the MEM_C2 configuration. For the rest of the configurations, we can observe that our solution is able to achieve similar performance when injection rates are below 2900 req/sec. Despite not being capable of outperforming larger configurations, our solution was able to keep response times below a defined threshold while keeping the amount of allocated memory below 1GB (except for the highest tested injection rate where 1GB was allocated to the VM). Therefore, saving 0.5 GB and 1.5GB when compared to MEM_C3 and MEM_C4, respectively, and injection rates are below 3000 req/sec. Also the memory utilization of our solution ranges from 46% up to 73% (resp. from lowest to highest tested injection rates), which shows that our solution is able to reduce resource waste (except for the base case, remaining configurations have memory utilization averages from 18% up to 41%). Regarding the time required to execute an adaptation action (if required), it ranged from 2 up to 3 minutes. The reason for such period is that an increment on the memory utilization does not necessarily imply an immediate SLO violation; hence, an increment on the allocated memory depends also on the RT predictor, which can delay the time taken to carry out the allocation/deallocation of memory, this without affecting SLOs.

These results can be considered as a clear indicator that our solution can accomplish its main objectives and is feasible to be used in more complex scenarios.

5 RELATED WORK

Most research systems and all commercial cloud solutions focus exclusively on horizontal scaling [9] [5] [11]. In the following, we discuss related approaches that support vertical scaling.

Farokhi et. al. [3] present a cloud controller for vertical memory elasticity. Their approach uses control theory to adapt the VM's allocated memory according to changes in the performance and resource usage of a VM. Specifically, the feedback variables for their control theory-based controllers are the application response time and memory utilization. Even though this approach is similar to ours in the pursued objectives, one key difference is the adaptation technique. Unlike our approach, their adaptation mechanism relies in control theory and weighted moving average (WMA) to achieve its objectives.

In [8] E. B. Lakew et. al. present performance models to achieve response time objectives by vertically scaling VMs. The models are based on queuing theory and use RLS filters to reduce the impact of measurement noise on their generated outputs. Unlike our approach, their work focuses on the adaptation of allocated vCPUs, rather than memory. The presented performance models

cannot be easily adapted to other resources. Moreover, adapting vCPUs implies that enough memory has to be guaranteed to the VM to avoid bottlenecks for the variable number of vCPUs. This can lead to resource waste either because the allocated memory is more than enough for the currently allocated vCPUs or because of vCPU underutilization. Also, our results show that using vCPU as a VM's performance indicator is not always accurate.

The approach proposed by Lei Lu et. al. in [10] is the one closest to our solution. Their solution makes use of auto-regressive-moving-average (ARMA) models to estimate the performance of applications based on the currently allocated resources to a VM. In their approach, the resources being considered are memory and CPU, which are also the targets of adaptation actions (vertical scaling). Unlike our solution, solution proposed by Lei Lu et. al. supports adjusting allocations both at the individual VM level and at the resource pool level (a VM collection), but it is dependant on the VMware scheduler.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have presented our RLS-based solution for the automatic adaptation of VMs running on clouds environments. Results show that our solution, when compared with VMs with larger and permanently-allocated computational resources, is able to maintain expected performance while reducing resource waste. This means that the objectives that our solution pursues were well addressed. Our proactive adaptation solution also has the advantage of adaptation actions (memory allocation) being almost immediately reflected after their execution. This characteristic is achieved through the utilization of the memory ballooning mechanism supported by recent Linux kernels. It also means that VMs do not have to be stopped at any time during the adaptation process; hence, reducing impact on deployed applications. Despite promising results obtained so far, the results presented here are only the starting point of our solution. In order to further improve our solution more experimentation is needed and possible better setup parameters (either for adaptation or predictors) have to be defined. As part of our future work we intend to perform experiments with real workload traces (e.g., FIFA 98 Worlcup traces), which has the objective of verifying not only the memory allocation functionality of our solution but also memory deallocation. Also, the addition of horizontal scaling capabilities to our approach is in process; the goal is to handle situations when the VM can no longer be vertically scaled. Finally, comparison with commercial solutions such as Openstack is also intended, so that we can verify the suitability of our solution in real world environments.

ACKNOWLEDGMENTS

We want to thank CONACYT for the scholarship provided to Carlos Ruiz.

REFERENCES

[1] Akindele A Bankole and Samuel A Ajila. 2013. Cloud client prediction models for cloud resource provisioning in a multitier web application environment. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*. IEEE, 156–161.

[2] Michael Borkowski, Stefan Schulte, and Christoph Hochreiner. 2016. Predicting Cloud Resource Utilization. In *Proceedings of the 9th International Conference on Utility and Cloud Computing (UCC '16)*. ACM, New York, NY, USA, 37–42. https://doi.org/10.1145/2996890.2996907

[3] Soodeh Farokhi, Pooyan Jamshidi, Ewnetu Bayuh Lakew, Ivona Brandic, and Erik Elmroth. 2016. A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach. *Future Generation Computer Systems* 65 (2016), 57 – 72. https://doi.org/10.1016/j.future.2016.05.028 Special Issue on Big Data in the Cloud.

[4] Stefan Frey, Simon Disch, Christoph Reich, Martin Knahl, and Nathan Clarke. 2015. Cloud Storage Prediction with Neural Networks. In *Cloud Computing 2015, The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization*. 52 – 56. https://www.thinkmind.org/index.php?view=article&articleid=cloud_computing_2015_3_10_20024

[5] Guilherme Galante and Luis Carlos E. de Bona. 2012. A Survey on Cloud Computing Elasticity. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing (UCC '12)*. IEEE Computer Society, Washington, DC, USA, 263–270. https://doi.org/10.1109/UCC.2012.30

[6] S. A. Ghauri and M. F. Sohail. 2013. System identification using LMS, NLMS and RLS. In *2013 IEEE Student Conference on Research and Developement*. 65–69. https://doi.org/10.1109/SCOReD.2013.7002542

[7] Simon Haykin. 1996. *Adaptive Filter Theory (3rd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[8] E. B. Lakew, C. Klein, F. Hernandez-Rodriguez, and E. Elmroth. 2014. Towards Faster Response Time Models for Vertical Elasticity. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. 560–565. https://doi.org/10.1109/UCC.2014.86

[9] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. 2014. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing* 12, 4 (2014), 559–592. https://doi.org/10.1007/s10723-014-9314-7

[10] L. Lu, X. Zhu, R. Griffith, P. Padala, A. Parikh, P. Shah, and E. Smirni. 2014. Application-driven dynamic vertical scaling of virtual machines in resource pools. In *2014 IEEE Network Operations and Management Symposium (NOMS)*. 1–9. https://doi.org/10.1109/NOMS.2014.6838238

[11] Athanasios Naskos, Anastasios Gounaris, and Spyros Sioutas. 2016. *Cloud Elasticity: A Survey*. Springer International Publishing, Cham, 151–167. https://doi.org/10.1007/978-3-319-29919-8_12