Performance-based Vertical Memory Elasticity

Soodeh Farokhi*, Pooyan Jamshidi†, Drazen Lucanin* and Ivona Brandic*
*Faculty of Informatics, Vienna University of Technology, Austria.
†Department of Computing, Imperial College London, United Kingdom.
Email: *{firstname.lastname}@tuwien.ac.at, †p.jamshidi@imperial.ac.uk

***Abstract*—Cloud computing offers the elasticity features by dynamically resizing the infrastructure in response to changes in workload demands to meet performance guarantees and minimize costs. In the last decade, a large body of work has been done in the area of horizontal elasticity, while only few research efforts addressed vertical elasticity. This paper develops a vertical elasticity controller for cloud-based applications using control theory principles to guarantee performance requirements by adjusting the memory allocation as a control knob. The novelty of our work lies on applying a controller synthesis technique by guaranteeing robustness and stability of the controlled system, using the application response time as a decision making criterion. The experimental results reveal that the controller is able to efficiently save at least 47% memory usage while keeping an acceptable user experience.**

I. Introduction

Modern web applications are getting more performance sensitive while they face unpredictable workloads. Therefore, resource auto-scaling is necessary not only to avoid application performance degradation but also to avoid acquiring resources which may not be used. In cloud computing, elasticity is a mean to decide the right amount of resources each application needs to avoid under- or over-provisioning. Two types of elasticity are defined: horizontal and vertical. While the former is the ability to provision and release virtual machines (VMs), the later is adjusting the capacity of a single VM to cope with runtime changes.

In this work, we design a controller for vertical memory elasticity of cloud-based applications (*mem-controller*) by adopting a controller synthesis technique introduced in [1] that guarantees the stability of a controlled system. The main motivation behind the choice of control theory in our work is to use this well-established theory for modeling and designing feedback loops to make the cloud-based applications self-adaptive and achieve a proper balance between faster reaction and better stability. Moreover, since the time to adjust memory at runtime is close to instantaneous, control theory is a good fit. We evaluated *mem-controller* based on an experimental study using RUBBoS, a cloud benchmark application, under Wikipedia and FIFA traces.

Different from the existing approaches in the domain of cloud vertical elasticity, our approach has several distinguishing aspects: (i) because of the special challenges in memory elasticity, research on this topic is scarce compared to other resources such as CPU [2]; (ii) among the work exploring memory elasticity, most of them [3] look at the data storage (DS) tier, as the effect of memory on retrieving data is clear; therefore, being concerned with the memory elasticity of the business logic (BL) tier (i.e., the tier hosting web server) has not yet been well investigated; (iii) we have looked at memory elasticity from a performance point of view instead of memory utilization as the most common used indicator

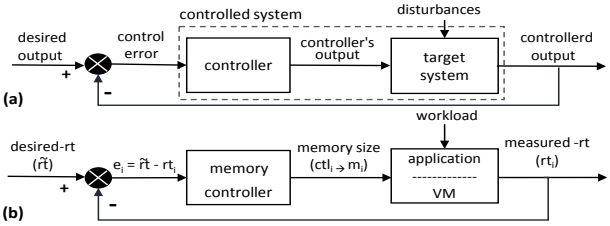


Figure. 1: (a) Standard feedback loop, (b) Our feedback loop. [5]

of the memory scarcity [4]. In other words, we consider application performance, i.e., response time (*RT*), as a decision making criterion to scale up or scale down the allocated memory of VMs. Notice that the proposed solution targets applications with dynamic memory requirements, and in which the performance bottleneck is memory, not CPU.

The contribution of the paper lies in developing and experimentally evaluating a controller by applying a controller synthesis technique and an error smoothing method. The controller goal is to realize the vertical memory elasticity and guarantee *RT* for cloud-based applications. The effect of memory elasticity of the BL tier on application performance is the main investigation in our work.

II. Memory Controller

In the context of control theory, a standard feedback control loop is as illustrated in Fig. 1[5] (a). In an equivalent feedback loop consisting of *mem-controller* shown in Fig. 1 (b), the target system is a cloud-based application deployed on a VM. The controller's output at each control iteration ctl_i is mapped to a memory size m_i to enable elasticity by scaling up or down the allocated VM memory. The parameters $\tilde{rt}$ and rt_i are the desired and measured *RT*, respectively. The control error e_i is the difference between these two values at each control iteration. The number of user requests (workload) and a change in the mix of different request types in the workload are considered as the disturbances, and as the controller cannot control the workload, it should change the application deployment environment in order to meet the desired *RT*.

The control law, as originally devised in [1], is presented in Eq. (1). As shown, the control output ctl_i is calculated based on its previous value ctl_{i-1} and a coefficient of the control error ($e_i = \tilde{rt} - rt_i$). The coefficient is based on the value of α, a model parameter, and *pole*. The parameter α is estimated at runtime based on the effect of ctl on rt. We apply the linear regression to capture this relationship.

$$ctl_i = ctl_{i-1} - \frac{1 - pole}{\alpha} \cdot e_i \quad (1)$$

The choice of *pole* determines the stability of the controlled system, and how fast it approaches to its equilibrium. The stability of the controller is ensured as long as $0 \leq pole < 1$[1].

[1] $pole = 0.99$ is used in our work.

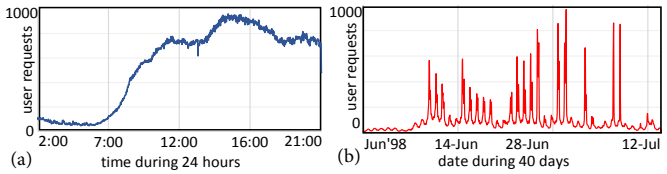


Figure. 2: Wikipedia trace (a); FIFA trace (b).

In order to develop a more stable and robust controller, we apply the *weighted moving average* error smoothing method used in time series analysis on calculating control error before using it in Eq. (1). At each control interval, *memory controller* tracks the $\tilde{rt}$ by rejecting the influence of workload fluctuation on the rt_i and withstands the control error e_i as long as it is insignificant. Te controller's output $ctl \in (0, 1)$ is mapped to a memory size $m_i \in [m_{min}, m_{max}]$ using Eq. (2).

$$m_i = ctl_i \cdot (m_{max} - m_{min}) + m_{min} \tag{2}$$

where m_{min} and m_{max} are the maximum and the minimum amount of VM memory sizes expressed by the number of memory units[2] m_{unit}, which are allowed to be allocated, m_i is the output of *mem-controller*.

III. Evaluation

For the evaluation, we compared the results of self-adaptive with two non-adaptive scenarios based on two provisioning policies: over- and under-provisioning in an experimental setup. We used a load generator tool with the ability to measure the *RT*, named `httpmon`[3] to measure *RT* at runtime. As shown in Fig. 2, while the used Wikipedia trace has a hourly incremental pattern, the FIFA trace includes sudden and temporal peaks. To facilitate the reproduction of our research, we released the source code (in `Python`) that prepares these traces[4]. The experiments were conducted on a single Linux Ubuntu 14.04 server, equipped with 16 processors and 32GB of memory. We used `KVM hypervisor` to create two VMs each for different tiers (BL and DS). Since the focus of this work is on memory elasticity of the VM hosting BL tier (VM1), we assigned a static number of CPU cores–4– to each VM. While the amount of allocated memory for VM1 is dynamic (512MB-6GB) and is managed by the controller, a static amount of memory–4GB–was assigned to the VM hosting DS tier (VM2). VM1 runs the application web server, `Apache 2.0`, and VM2 runs the application database, `MySQL`.

For the evaluation metrics, we consider: (i) the average of measured *RT*s; (ii) the average number of memory units used over time, which determines the variable part of the ownership cost. This memory saving is beneficial for the resource provider, since the physical host can take advantage of that memory reduction and uses it for other VMs; (iii) violation rate throughout the experiment. The goal is to meet the desired *RT*, which is set to $\tilde{rt} = 600ms$ and $300ms$ in the experiments associated with the Wikipedia and the FIFA traces, respectively, while keeping memory usage and violation rate as low as possible. A visual summary of the aggregated results is depicted in diagrams of Fig. 3 and is discussed briefly in the following sections.

Wikipedia trace. As shown in Fig. 3 (a), in comparison with the over-provisioning policy, the controller has acquired less memory, so decreasing the resource cost by 47 percent (51 vs. 96 unit). In comparison with the under-provisioning policy, *mem-controller* is significantly better in terms of *RT* (490ms vs. 2109ms), giving applications' owners a better chance to guarantee the performance metric. The percentile of time in which violation has accrued during the experiment is shown as *RT* violation rate in Fig. 3. In an ideal situation, a controller should be able to keep the violation rate close to zero. As depicted, although the controller was able to keep the total average of measured *RT* less than the desired *RT* (490ms vs. 600ms), at 18% of time, the violation was accrued.

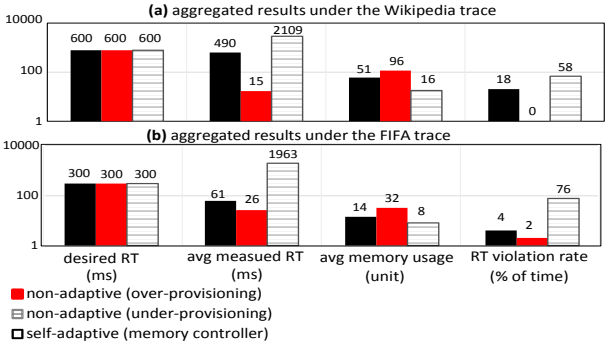


Figure. 3: Results under the Wikipedia (a), and the FIFA (b) traces.

FIFA trace. For the results shown in Fig. 3 (b), the situation is much better. In comparison with the over-provisioning policy, the controller has allocated less memory, so consequentially decreasing the memory usage by 57% (14 vs. 32 unit). In comparison with the under-provisioning policy, it behaves better in terms of the average of measured *RT* (61ms vs. 1963ms). In this workload trace, the violation rate is 4%.

IV. Conclusion

In this paper, we developed a vertical memory elasticity controller by using a controller synthesis technique and error smoothing method. The controller adjusts the allocated VM memory to boost performance by taking the application response time as an indicator of the memory scarcity. The evaluation results show the benefit of the controller to save the memory usage (cost) by 47% in case of the Wikipedia, and 57% in case of FIFA traces in comparison with the over-provisioning policy, while satisfying the desired response time in all scenarios. Such a controller can be used to make a cloud-based application self-adaptive (memory-wise) and to guarantee the desired performance while decreasing the cost for the application owner.

Acknowledgment

This work was supported by the HALEY project, and the WWTF through the PROSEED grant.

References

[1] A. Filieri, H. Hoffmann, and M. Maggio, "Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees," in *ICSE*, 2014, pp. 299–310.

[2] S. Spinner, S. Kounev, X. Zhu, L. Lu, M. Uysal, A. Holler, and R. Griffith, "Runtime vertical scaling of virtualized applications via online model estimation," in *SASO*, 2014, pp. 157–166.

[3] L. Lu, X. Zhu, R. Griffith, P. Padala, A. Parikh, P. Shah, and E. Smirni, "Application-driven dynamic vertical scaling of virtual machines in resource pools," in *NOMS*, 2014, pp. 1–9.

[4] A. Baruchi and E. T. Midorikawa, "A survey analysis of memory elasticity techniques," in *Euro-Par*, 2011, pp. 681–688.

[5] S. Farokhi, P. Jamshidi, I. Brandic, and E. Elmroth, "Self-adaptation challenges for cloud-based applications: A control theoretic perspective," in *Feedback Computing*, 2015.

[2] memory unit is a discrete block of memory, e.g., 64MB in this work.

[3] https://github.com/cloud-control/httpmon

[4] Wikipedia: http://goo.gl/iy36Pg and FIFA: http://goo.gl/iUGF6O

Performance-based Vertical Memory Elasticity

Soodeh Farokhi[1], **Pooyan Jamshidi**[2], **Drazen Lucanin**[1], and **Ivona Brandic**[1]

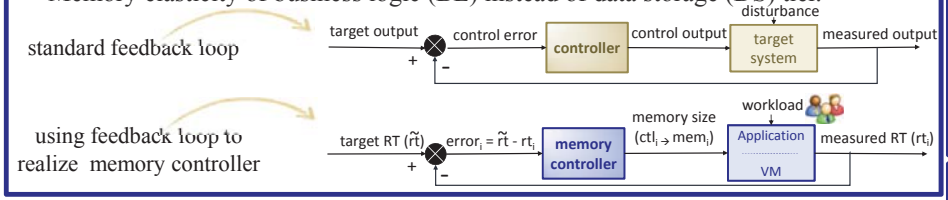
[1]Faculty of Informatics, Vienna University of Technology, Austria. [2]Department of Computing, Imperial College London, UK.

Introduction

Cloud computing offers the elasticity features by dynamically resizing the infrastructure in response to changes in workload demands to meet performance guarantees and minimize costs. In the last decade, a large body of work has been done in the area of horizontal elasticity, and among the existing vertical elasticity approaches only a few research efforts addressed **memory vertical elasticity**. This work develops a vertical elasticity controller for **cloud-based applications** using **control theory** principles to guarantee performance requirements by adjusting the memory allocation as a control knob and the **application response time** (RT) as a decision making criterion.

Motivation

- Why using control theory in this work?
  - adopting feedback loop to make application self-adaptive (memory-wise)
    - allocated memory as a control knob
    - application RT as a feedback (decision making criterion)
  - achieving a proper balance between faster reaction and better stability.
  - time to adjust memory at runtime is almost instantaneous
- Memory elasticity of business logic (BL) instead of data storage (DS) tier.



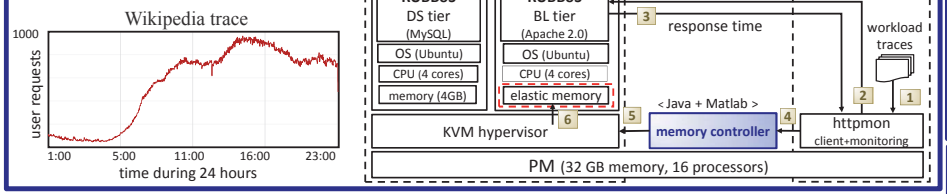
Memory Controller

- Adopting a control synthesis technique [1] which includes two phases:
  - learning phase to capture system model α (using linear regression)
  - control phase to control the response time based on the control formula [1]
    - using weighted moving average as an error smoothing technique

calculating control signal using control law: $ctl_i = ctl_{i-1} - \frac{1 - pole}{\alpha} error_i$

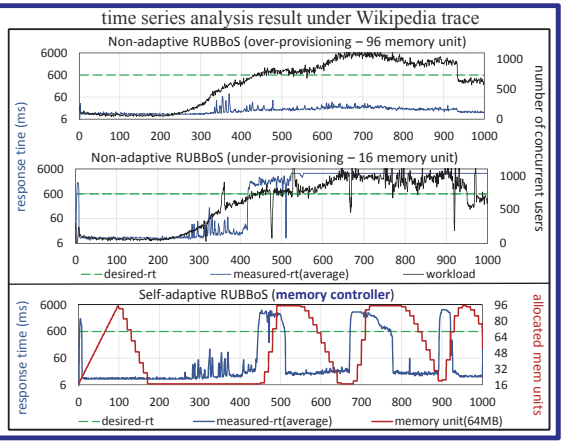
mapping control signal to memory: $mem_i = ctl_i\,(mem_{max} - memm_{in}) + memm_{in}$

experimental evaluation setup



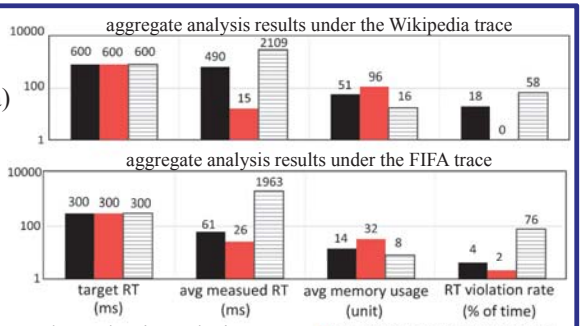
Experimental Evaluation

- Experimental Setup:
  - managing VMs by KVM hypervisor
  - using RUBBoS as a benchmark app.
  - using Wikipedia and FIFA traces
- Comparing self-adaptive results with
  - over-provisioning
  - under-provisioning
- Aggregate analysis evaluation metrics
  - SLA (average response time)
  - memory-usage
  - violation rate (% in time)



Summary of Results

- Aggregated results
  - less memory by up 47% (Wikipedia) (SLA violation: 18% of time)
  - less memory by up to 57% (FIFA) (SLA violation: 4% of time)
  - Total average of RTs satisfy SLA



Conclusion

Memory controller was designed by a control synthesis technique

- taking allocating memory as control knob
- taking response time as an indicator of the memory scarcity
- guaranteeing target performance while decreasing resource (memory) cost

- Using control theory to make cloud-based applications self-adaptive (memory-wise)

Future Work

- Focusing on multiple resources (CPU & memory) controllers.
- Building adaptive system model by using Recursive Least Square (RLS) filtering technique.

References

[1] A. Filieri, H. Hoffmann, and M. Maggio, "Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees," ICSE 2014, pp. 299-310.