EFFECTIVE RESOURCE ALLOCATION USING AUTONOMIC ELASTICITY SCALING APPROACHES IN CLOUD COMPUTING

Anush Neelakantam
Department of Computer Science
George Mason University

Agenda

- ABSTRACT
- INTRODUCTION
- ELASTICITY-DEFINITION AND QUANTIFICATION
- HYBRID CONTROLLER FOR VERTICAL AND HORIZONTAL SCALING
- AUTONOMIC MEMORY AND CPU RESOURCE ALLOCATOR
- PRESS, PRedictive Elastic ReSource Scaling
- Lightweight Resource Scaling for Cloud Applications
- EXPERIMENTAL EVALUATIONS
- OPEN PROBLEMS AND FUTURE WORK
- REFERENCES

Abstract

- Elastic provisioning is gaining a lot of popularity in cloud platforms.
- Elasticity - Ability of the system to adapt to workload changes by adjusting the resources in an autonomic manner while maintaining Service Level Objectives (SLOs)
- Not much work done on measuring the elasticity value.
- Initially, we study a mathematical approach to quantify the elasticity value of a system.
- Proceed to study two hybrid controllers. First controller implements horizontal and vertical scaling of resources. The second controller studies a unique approach to vary the memory and the CPU cores to perform the vertical scaling of resources.
- Extend the research to study a proactive controller and a lightweight resource scaling approach
- End the paper with open problems and a few solutions.

INTRODUCTION

- Previous method used by organizations: Hard drives
- Drawbacks: Low availability and constant updates

- Introduced Cloud Computing in which users access data over the internet and also host applications using third party resources such as virtual machines (VMs)

- Advantages:
- Multiple request processing
- Pay-per-use model
- High accessibility

Elasticity-Definition and Quantification

- Re-define elasticity:

  Just-in-need state T_j

  Overprovisioning state T_u

  Under provisioning state T_u

- Definition: The total time spent by the system in the just-in-need states.

$$E = T_j/T_m$$

$$E = 1- T_o/T_m – T_u/T_m$$

If we assume P_j, P_o, P_u to be the accumulated probabilities for the states then we arrive at the equation:

$$E = P_j = 1 – P_o – P_u$$

Elasticity-Definition and Quantification

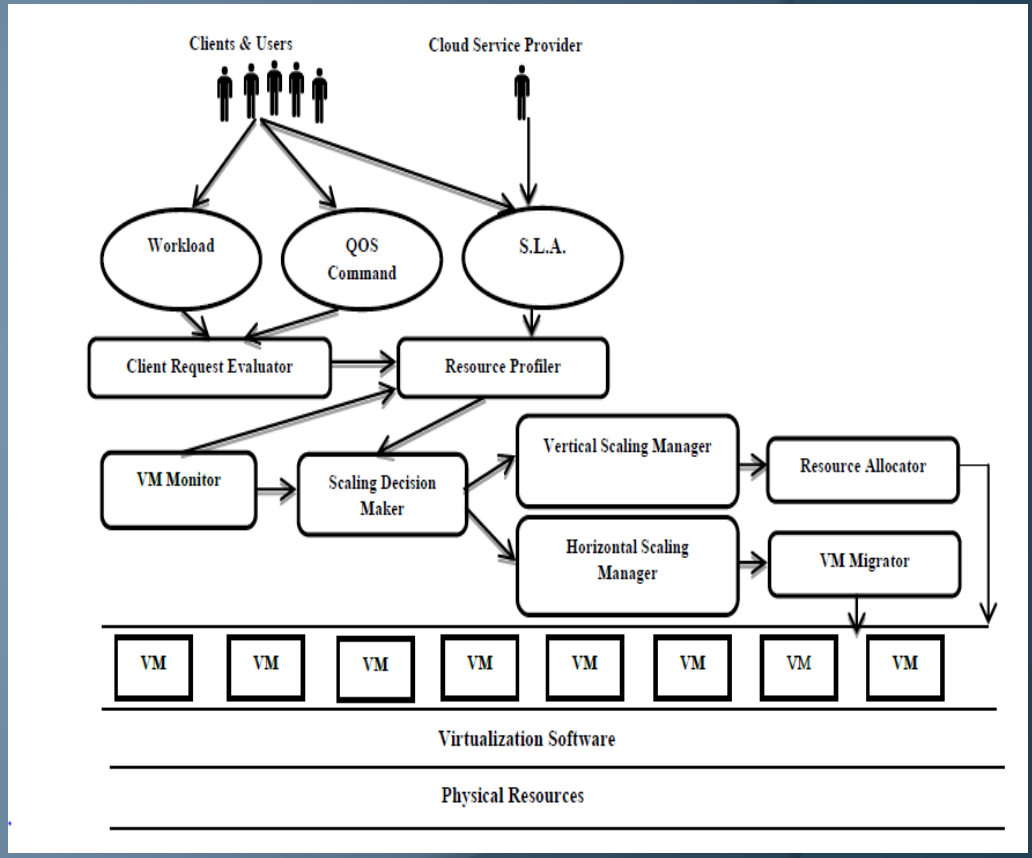
- Can use the previous equation to compute the elasticity of a cloud platform by using a CTMC model.
- A continuous-time Markov chain (CTMC) is a continuous time, discrete state Markov process which basically transitions the system from one state to any one of the neighboring states.
- The input and output parameters of our CTMC model are summarized in the following. Input: The request arrival rate is C, the service rate is μ, the virtual machine start-up rate is α, and the virtual machine shut-off rate is β . (In addition, the definitions of "just-in-need," "under provisioning," and "overprovisioning" states should also be included.)
- outputs the accumulated under-provisioning state probability (P_u) and the accumulated overprovisioning state probability (P_o) as:

$$P_{\mathrm{u}} = \sum_{i=1}^{m} \sum_{j=3i+1}^{m+1} P_{i,j},$$

$$P_{\mathrm{o}} = \sum_{i=2}^{m} \sum_{j=0}^{i} P_{i,j},$$

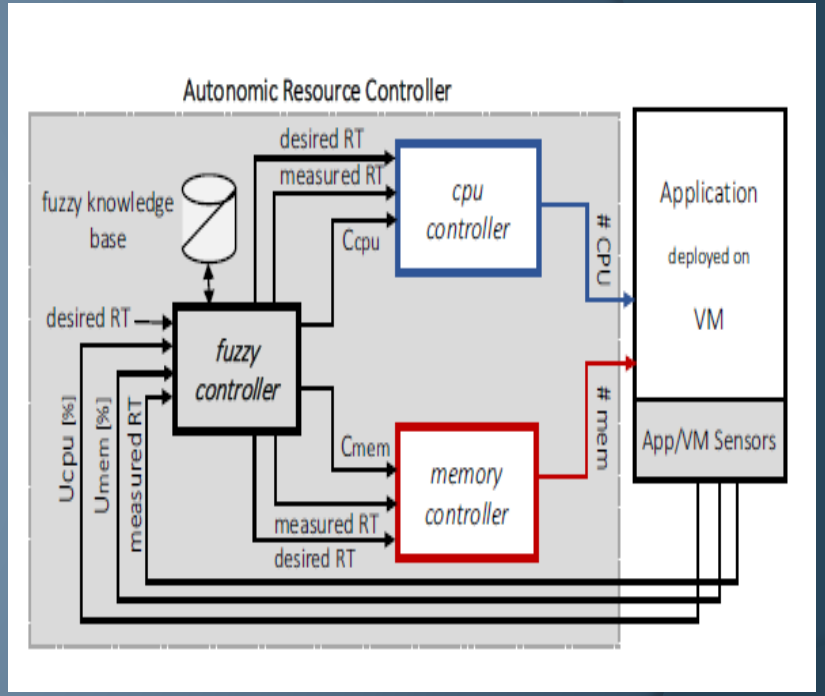
HYBRID CONTROLLER FOR VERTICAL AND HORIZONTAL SCALING

- Client request evaluator and a VM monitor to evaluate the future resource demand of the system and study the resource usage respectively.
- The resource profiler utilizes the two values obtained from the client request evaluator and VM monitor to coordinate with the scaling decision maker.
- The scaling decision maker performs the scaling decisions.
- Approach:
- When the workload experiences a sudden increase, the free resources in each VM are utilized to maintain the SLOs
- If the free resources are also not sufficient enough to handle the workload, the horizontal scaling manager spawns additional VMs and redistributes the data using a load balancer.



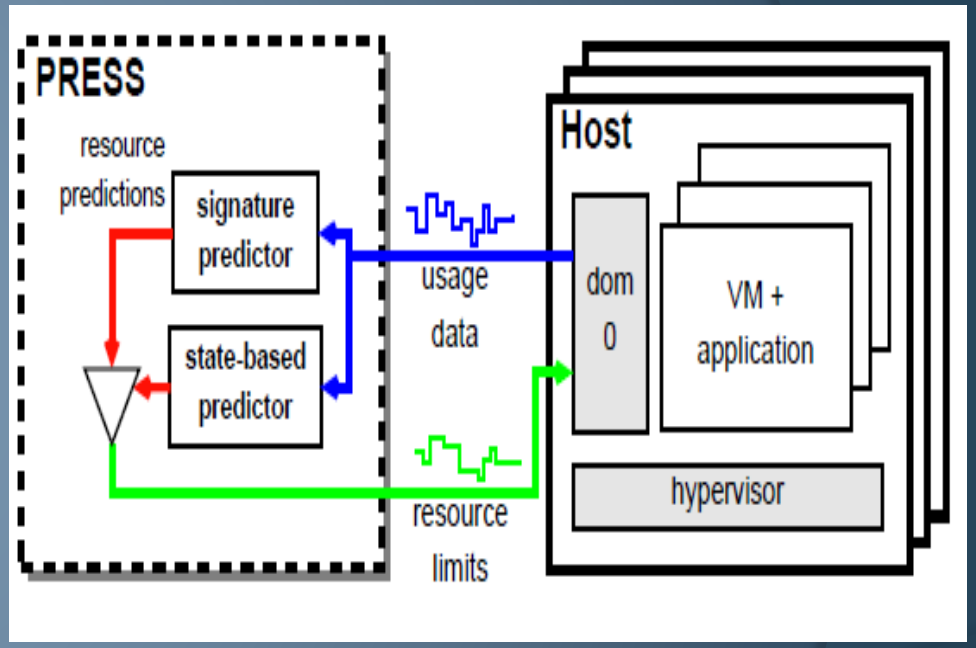
AUTONOMIC MEMORY AND CPU RESOURCE ALLOCATOR

- The autonomic resource controller follows a MAPE-K loop based on self-adaptive systems
- The fuzzy controller reads in the average CPU and memory utilization and determines the impact that CPU and memory have on the system.
- The analysis phase involves the memory and cpu controllers deciding the amount of individual resources needed to satisfy the performance constraints of the application.
- Finally, during the planning and execute phases, the hypervisor is instructed to add or remove the number of resources computed by the memory and cpu controllers.



PRESS, PRedictive Elastic ReSource Scaling

- Proactive prediction based approach that extracts fine-grained patterns of the resource demands in a system and adjusts the resources accordingly
- PRESS uses the signature based predictor to predict the resource demands of workloads having repeating patterns
- State-driven resource demand predictor is used if there is no recurring pattern in the workload.
- Approach:
- Initially, PRESS sets the CPU utilization to a maximum and after a few resource demand values are noted, PRESS uses either of the two predictors based on the repetitive or non-repetitive nature of the workload.
- Once confident of the predictions made, PRESS begins to scale the resources accordingly. To ensure accuracy of the predictions, PRESS constantly updates the predictors using recent resource demand values



Lightweight Resource Scaling for Cloud Applications

- Addition/removal of resources dynamically generally incurs additional overhead to redistribute the data among the newly added servers
- In order to reduce the aforementioned scaling overhead, the authors discuss the implementation of a lightweight resource level scaling approach which ensures cost-efficient resource allocation.
- The authors implement the lightweight scaling of resources using two major approaches.
- The first approach, known as the self-healing scaling is used when two servers run on a single physical machine. The self-healing scaling approach aims to use to idle resources of one instance to support the overloaded instance on the PM.
- The second approach, known as the resource-level scaling, aims to use the available resources on the PM to satisfy the resource demands of a VM executing on it.

EXPERIMENTAL EVALUATIONS

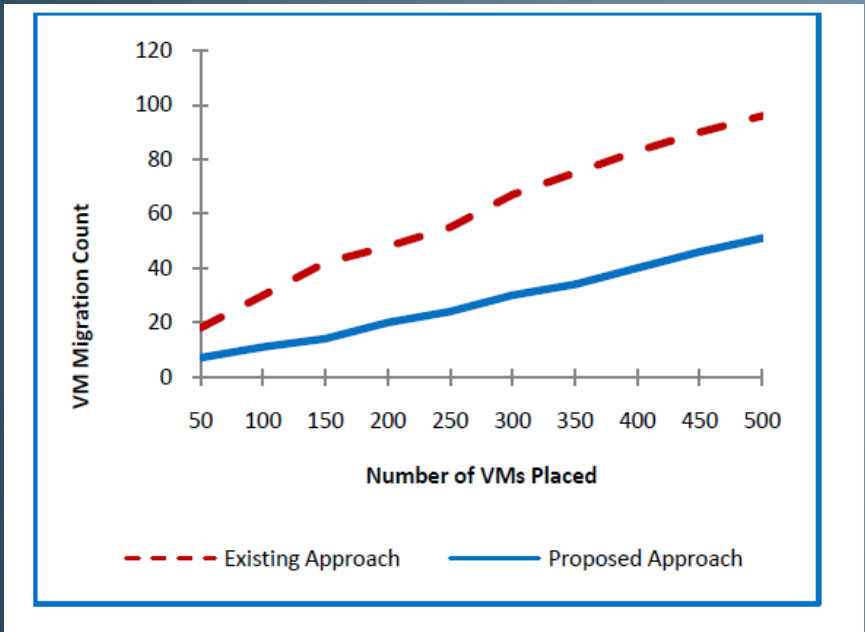
- Evaluations for the elasticity quantifying approach:
- compare the CTMC model against a system called Cloud Elasticity Value

- Results indicate that the values of elasticity predicted by the CTMC model were very similar to that of the Cloud Elasticity Value simulator. A max difference of 0.8% existed.

- conducted experiments on a cloud computing environment called LuCloud. A max difference of 0.8% existed.

EXPERIMENTAL EVALUATIONS

- Evaluations for the hybrid elasticity controller:



- Evaluations for the fuzzy logic controller:
- NFC over-provisioned memory and CPU, while FC allocated resources as required.
- Aggregate analysis:
- Results indicate that under the RUBiS open system model, the memory allocated by FC was 60.76% less than the NFC approach while ensuring high stability.
- When using the Olio benchmark, the number of CPU cores allocated by the FC was 56.51% less than that of the CPU cores allocated by the NFC approach while again maintaining high stability.

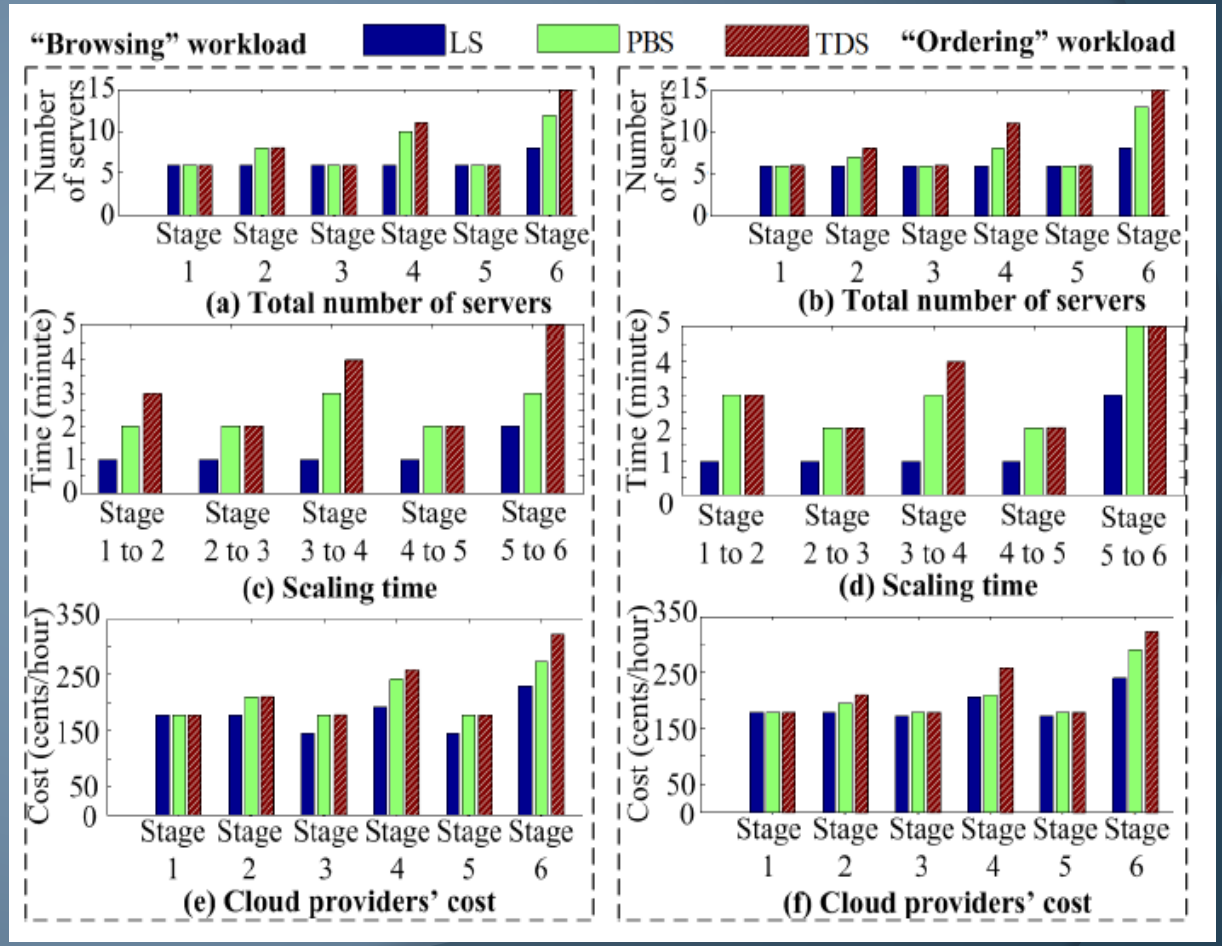
EXPERIMENTAL EVALUATIONS

- Evaluations for PRESS:
- The first experiment performed in the evaluations studied the effectiveness of the predictors in predicting the actual resource demand.
- PRESS almost always outperformed other predictors like auto-correlation, mean-32 and max-4320.
- The second experiment studied the individual prediction errors made by individual predictors for the RUBiS web server workload
- Results indicated that the PRESS method only made a small investment of 9% in CPU overhead for predictions and delivered up to 70% resource savings when compared to static allocation

EXPERIMENTAL EVALUATIONS

- Evaluations for Lightweight scaling
- Compared the LS scaling algorithm against Policy-based scaling (PBS) and the Tier-Dividing Scaling (TDS) algorithms.

- The results indicated that the LS algorithm had the least amount of increase in the number of servers, used the least amount of scaling time and saved significant amounts of service provider cost by removing idle resources



OPEN PROBLEMS AND FUTURE WORK

- A good topic to further discuss would be the measurement of the cost incurred due to VM migration and also the network bandwidth saved due to use of the hybrid controller.
- An initial approach to tackle this problem would be to integrate the system with modern bandwidth monitoring tools such as the Solarwinds RTBM, Networx, BitMeter II, BWMonitor and Spiceworks.

- Problem: Tedious offline training phases
- Solution: An initial approach towards the online training of elasticity controllers would be to implement support vector machines along with the fuzzy logic scheme. Using SVMs along with the continuously growing database of fuzzy rules would update the controller dynamically and avoid the offline training phase.

- Issue: Resource contention in various tiers
- Solution: Implement elasticity in all the tiers and not just the service layer.

REFERENCES

- Ai, Wei, Kenli Li, Shenglin Lan, Fan Zhang, Jing Mei, Keqin Li, and Rajkumar Buyya. “On Elasticity Measurement in Cloud Computing.” *Scientific Programming* 2016 (2016): 1–13. https://doi.org/10.1155/2016/7519507.

- Shelar, Madhukar, Shirish Sane, and Vilas Kharat. “Enhancing Performance of Applications in Cloud Using Hybrid Scaling Technique.” *International Journal of Computer Applications* 143, no. 2 (June 17, 2016): 43–48. https://doi.org/10.5120/ijca2016910027.

- Farokhi, Soodeh, Ewnetu Bayuh Lakew, Cristian Klein, Ivona Brandic, and Erik Elmroth. “Coordinating CPU and Memory Elasticity Controllers to Meet Service Response Time Constraints,”69–80.IEEE, 2015.https://doi.org/10.1109/ICCAC.2015.20.

- Gong, Zhenhuan, Xiaohui Gu, and John Wilkes. “Press: Predictive Elastic Resource Scaling for Cloud Systems.” In *Network and Service Management (CNSM), 2010 International Conference On*, 9–16. Ieee, 2010.

- Han, Rui, Li Guo, Moustafa M. Ghanem, and Yike Guo. “Lightweight Resource Scaling for Cloud Applications.” In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium On*, 644–651. IEEE, 2012.