2013 IEEE Third International Conference on Cloud and Green Computing

Virtual Machine Resource Allocation in Cloud Computing via Multi-Agent Fuzzy Control

Dorian Minarolli and Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
{*minarolli, freisleb*}*@informatik.uni-marburg.de*

***Abstract*—Dynamic resource (re)-allocation for virtual machines in cloud computing is important to guarantee application performance and to reduce operating costs. The problem is to find an adequate trade-off between these two conflicting goals. An approach is presented to support the Virtual Machine Monitor in performing resource allocation of VMs running on a physical machine of a cloud provider by expressing the two objectives in a utility function and optimizing this function using fuzzy control. To potentially work for an increased number of virtual machines, a multi-agent fuzzy controller is realized where each agent optimizes its own local utility function. The multi-agent fuzzy controller is empirically compared to a centralized fuzzy controller and an adaptive optimal control approach. Experimental results show the effectiveness of the multi-agent fuzzy controller in finding an adequate trade-off between performance and cost.**

***Keywords*-cloud computing; fuzzy control; utility function; multi-agent**

I. Introduction

Cloud computing offers improved manageability and reduced operating costs of computational resources to providers. This is made possible by virtualization solutions such as Xen [2] or VMware [17]. They provide flexible resource provisioning mechanisms such as dynamic allocation of resources to virtual machines (VMs). Since application workloads change over time, the optimization problem to be solved is how resources should be allocated dynamically to keep application performance according to service level agreements (SLAs) while reducing operating costs.

In this paper, an approach is presented to support the Virtual Machine Monitor (VMM) in allocating resources to VMs running on a physical machine of the cloud provider. The two conflicting goals of maintaining application performance and reducing operating costs are expressed in a utility function that represents the profit of a cloud provider. The problem to be solved is optimizing the utility function, and the solution proposed in this paper is based on fuzzy control to optimize it. To provide scalability for a potentially large number of VMs, a multi-agent version of the fuzzy controller is presented where each agent is responsible for resource allocation of a single VM in parallel with other agents, while an agent coordinator redistributes the maximum amount of resources that can be used by all agent VMs.

We focus on fine-grained dynamic allocation of VM resources locally on each physical machine of a cloud provider and consider CPU and memory as resources to be managed. Resources are allocated through the mechanisms offered by the used virtual machine technology for changing the CPU share and the memory allocation at runtime. Since the used virtualization technology does not adequately support disk I/O bandwith allocation, we do not consider it in this paper. However, the proposed approach is principally designed to also include disk I/O bandwidth allocation. Fuzzy control is used to maximize a global utility function (a utility function over all VMs) using a hill-climbing heuristic implemented as fuzzy rules. In the multi-agent version, the global utility function is divided into local utility functions that are optimized separately by each agent. The advantage of using fuzzy control is that it does not require any mathematical model to be built beforehand to control VM resources. The novelty of our work consists in developing a fuzzy control approach for utility function optimization and supporting a potentially large number of VMs by developing a multi-agent fuzzy controller.

The performance of the proposed approach is compared to an adaptive optimal control approach developed by Magnus *et al.* [9] that we adapted for VM resource allocation in our context. It represents an advanced control theory approach and is one of the state-of-the-art techniques used recently for virtualized resources [11]. Experimental results show the effectiveness of the multi-agent fuzzy control approach achieving better utility values than the approach of Magnus *et al.* [9] and the centralized fuzzy controller.

The paper is organized as follows. Section II discusses related work. Section III describes the VM resource allocation framework. Section IV discusses the adaptive optimal control approach. Section V introduces the centralized fuzzy control approach, and Section VI its multi-agent version. Section VII describes experimental results. Section VIII concludes the paper and outlines areas for future work.

II. Related Work

Several approaches to dynamic resource management have been proposed in the literature. One group of work [5], [3] is based on queuing theory to model performance

978-0-7695-5114-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CGC.2013.35

and manage resources. These approaches are based on mathematical models that have difficulties to capture the complex relationship between resource allocation and application performance in shared virtual infrastructures. Other works [18], [11], [9] use control theory for resource management. They are based on linear models and require some effort to be applied in complex virtualized environments. Our approach differs from all of the above works, because we use fuzzy control without the need to build a model beforehand.

Another group of work [19],[7],[20] is based on supervised machine learning techniques to manage application performance. For example, Kundu *et al.* [7] use a neural network to model application performance in virtualized shared infrastructures using multiple resource knobs. Wildstrom *et al.* [20] also use machine learning for modeling performance based on low level metrics to find the best configuration to increase the throughput. In contrast to all machine learning techniques mentioned above, our approach is based on fuzzy control.

In other works, reinforcement learning [16], [13] is used for resource configuration. An agent converges to the best policy by trying different actions in the environment and taking the ones that give increased reward values. One of the difficulties of this approach is its long convergence time, making it impractical for production systems.

There are works that use fuzzy control for profit and performance management. Diao *et al.* [4] use fuzzy control to optimize profit, but they manage only specific applications in a non-virtualized environment using only a single control knob. Rao *et al.* [14] manage multiple objectives in virtualized environments using fuzzy control, but they apply fuzzy control to control objectives such as performance and costs separately and resolve conflicts through another scheduler layer, while we express conflicting objectives in a utility function and apply fuzzy control to optimize it. Furthermore, in contrast to the above fuzzy control techniques, we have developed a multi-agent version of a fuzzy controller that is suitable for large-scale multiple VM environments.

There are works [15], [8], [10] on distributed resource allocation and utility function optimization. In general, they focus on developing mathematical frameworks and depend on mathematical models of a utility function on resources allocated for distributed utility optimization. We consider distributed utility optimization and resource allocation in real application environments using a hill-climbing heuristic without using any mathematical model.

III. Virtual Machine Resource Allocation Framework

The problem considered is how resources of a physical machine of a cloud provider should be (re)-allocated to VMs dynamically in response to workload changes to keep the performance according to the SLAs while reducing the operating costs. An SLA is a contract between a consumer and a cloud provider, and in our case it is represented by the utility function that describes the monetary value that a consumer pays for obtaining the desired performance level. The operating cost includes any cost from management costs to power costs of the physical machine. In our case, we use the average amount of resources allocated to all VMs as an approximation of the operating costs. We solve the problem of finding the right trade-off of performance and operating costs by optimizing the following global utility function:

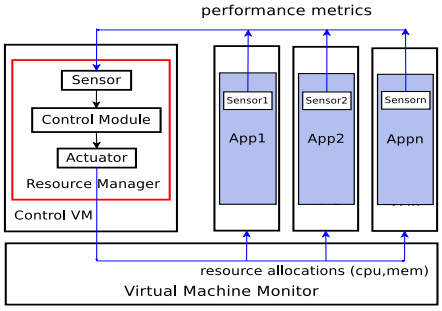


Figure 1. Resource manager architecture

$$U = \delta * (\alpha \frac{\sum_{i=1}^{n} V_i}{n} - \beta C) \quad (1)$$

U represents the average profit the cloud provider gets from one physical machine during a control interval, n is number of VMs, V_i is the performance utility that represents the monetary value paid by the consumer to the cloud provider for getting a certain performance level from VM_i during a control interval, and C represents the operating cost of the physical machine in a control interval. The coefficients α and β are used to give more priority to one objective over the other. δ is a constant to make the utility value larger than 1 for display convenience. To measure application performance, we use the application heartbeat framework [6]. Although we use this technique to measure application performance, our approach is independent of any performance measuring method, since we use the normalized performance calculated by dividing the actual performance by the desired performance level in an SLA contract. V_i is a simple linear function of normalized performance n_perf:

$$V_i = \begin{cases} n\_perf & \text{if } n\_perf < 1, \\ 1 & \text{if } n\_perf >= 1. \end{cases} \quad (2)$$

The operating cost C is the average of the sum of resource allocation amounts of two resource types, CPU and memory:

$$C = \frac{\sum_{i=1}^{n} cpu_i + \sum_{i=1}^{n} mem_i}{2} \quad (3)$$

where cpu_i and mem_i are portions of CPU and memory capacity given to VM_i expressed as numbers in the interval [0, 1], obeying the constraints that the total sum should be less than 1, the resource capacity. We have implemented our approach as a resource manager responsible for resource

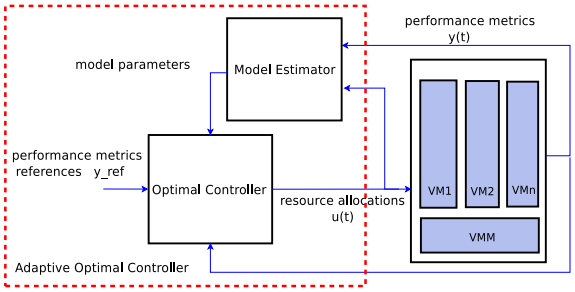


Figure 2. Adaptive optimal controller

Algorithm 1: Adaptive optimal controller algorithm

```
1  apply few predefined samples to build an initial model
2  while true do
3      sleep(30sec)
4      obtain performance measurements y(t)
5      estimate model parameters using RLS
6      calculate resource alloc u(t) through the control law
7      if sum of resource allocations greater than capacity then
8          redistribute resources in the same proportions to be
           lower than capacity
9      end
10     apply resource alloc u(t)
11 end
```

allocation of VMs running on a physical machine. Its architecture shown in Fig. 1 consists of three components: 1) sensor, 2) actuator, 3) control module. Sensors gather performance metrics in each interval, normalize them and pass them to the main sensor in the resource manager. The actuator gets resource allocations from the control module and applies them via the VMM. The control module implements the resource allocation approach. In each interval, it decides which resource allocation should be applied in the next interval.

IV. Adaptive Optimal Control

Adaptive optimal control is an advanced design approach to build feedback controllers. The method of adaptive optimal control used in this paper is based on the work of Magnus et al. [9], but we have adapted to a virtualized environment where VM CPU and memory shares are used as controller outputs instead of shares for scheduling requests. The architecture of the adaptive optimal controller is shown in Fig. 2. It consists of two components: 1) *model estimator* and 2) *optimal controller*.

The model estimator computes a linear autoregressive-moving-average Multi-Input Multi-Output (MIMO) model of the relationship between multiple resource allocations and multiple application performance metrics. Since workload dynamics can change overtime, this linear model may become invalid and thus should be recomputed. For this purpose, a recursive least squares (RLS) method is used to recompute the time-varying model parameters online.

The optimal controller aims to find resource allocations that minimize a quadratic cost function from which a control law can be derived to calculate resource allocations [9].

Algorithm 1 shows the main steps of the adaptive optimal controller. In the algorithm, the control interval between resource allocation decisions is set to 30 seconds through sleep() which puts the control module to sleep. This value is chosen as a trade-off between a low value with noisy measures and a high value of slow reaction.

V. Centralized Fuzzy Control

Fuzzy control [12] provides a way to design a controller based on heuristic knowledge needed to control a dynamic process. Our fuzzy controller is shown in Fig. 3. The output of the VM environment is the utility value $u(t)$ to be maximized. The change of the utility value $du(t)$ between two intervals is the input of the fuzzy controller. The output of the fuzzy controller is the allocation change $da(t+1)$ determining whether a resource should be increased or decreased in the next interval. This is also applied as the input to the fuzzy controller itself as the current resource allocation change $da(t)$ to be used for the next decision.

Based on the change of the utility and the change of allocation in the previous interval, the controller determines the change of allocation for the next interval. Since the fuzzy controller internally works with linguistic variables and linguistic values, the fuzzyfication module performs the conversion from numeric values measured from outside the fuzzy controller to the corresponding linguistic values (e.g., *poslarge*, a value that is positive and large), while the defuzzyfication module performs the reverse conversion. In the rule base, a set of condition-action rules is stored that implements the heuristic to optimize the utility function. The inference mechanism based on the linguistic input values selects the rules that apply and produces linguistic output values. The conversion in the fuzzyfication and defuzzyfication modules from numeric values to linguistic values is achieved by a membership function. This function maps a numerical value to certainty levels, a number in the interval [0,1] (0 completely uncertain, 1 completely certain), for different linguistic values it corresponds to. In Fig. 4(a), our

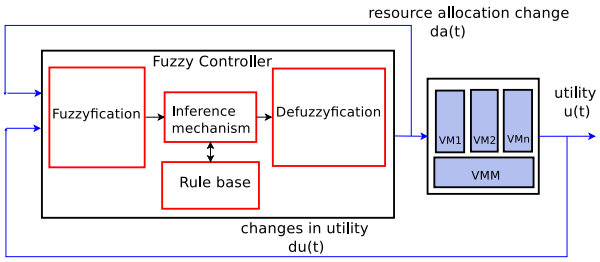
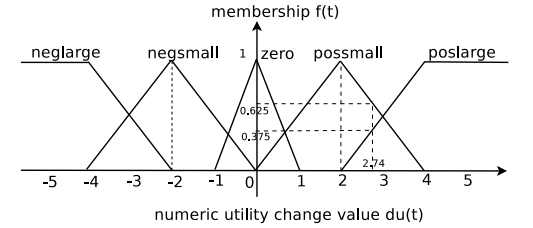


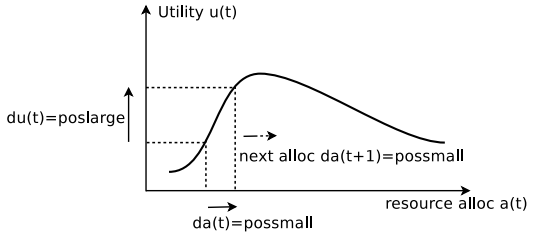
Figure 3. Fuzzy controller

membership function that converts a numeric utility change to linguistic values with different certainties is shown, where the x-axis represents numeric values and the y-axis certainty values. We use five linguistic values to represent utility changes. The membership function for resource allocation changes is not shown for space reasons, but it has almost the same form, and for the input it is used to convert numeric resource allocation changes to linguistic values, and for the output linguistic values to numeric resource allocation changes.

Our fuzzy controller maximizes the utility value by a set of rules based on a hill-climbing heuristic. Fig. 4(b) shows how the utility value changes by changing the allocation of a resource for a VM. If we increase the resource allocation in the current interval by a small amount and get a positive large increase of the utility value, we are on the left side of the hill and the slope of the curve is high. Thus, to maximize the utility, we should continue to increase the allocation in the next interval by a small amount in order not to overpass the top of hill. The idea of using hill-climbing to maximize the utility is based on work of Diao *et al.* [4].



(a) Utility change membership function



(b) Hill-climbing heuristic for one resource

Figure 4. Membership function and hill-climbing heuristic

Table I
HILL-CLIMBING HEURISTIC RULE TABLE

da(t+1)		du(t) 0	1	2	3	4
da(t)	0	1	0	2	3	3
	1	0	1	2	4	4
	2	2	2	2	2	2
	3	4	3	2	0	0
	4	3	4	2	1	1

Similar rules are created for all situations, as shown in Table I. This rule set has been created by experimental testing, and the results indicate that it performs quite well. The following mapping between numbers and linguistic values is used: *0-possmall*, *1-poslarge*, *2-zero*, *3-negsmall*, *4-neglarge*. These rules tell how to maximize utility for one resource of a VM, and since utility is a function of multiple resources and multiple VMs, we maximize the utility function by applying these rules for each resource and each VM one by one as shown in Algorithm 2 without lines 27-29, which are used only for multi-agent fuzzy control. The algorithm starts with minimum resource allocations and takes a first round, in lines 3-14, of only increasing allocations by applying fuzzy rules until it reaches the optimal utility. Then, it takes a second round, in lines 15-26, of only decreasing allocations until a possibly new optimal utility is reached. This repeated switching between the two rounds allows the controller to adapt to changes in optimal utility due to workload changes.

Algorithm 2: Fuzzy control algorithm

```
1  set resource allocations of all VMs to minimum levels
2  while true do
3      for i ← 1 to numberOfVMs do
4          for j ← 1 to numberOfResources do
5              nextalloc = possmall // increase resource
6              repeat
7                  apply nextalloc to resource j of VMi
8                  sleep(30sec)
9                  based on utility change apply fuzzy rules
                   and get nextalloc
10             until utility change zero or negative ;
11             apply nextalloc to resource j of VMi
12             sleep(30sec)
13         end
14     end
15     for i ← 1 to numberOfVMs do
16         for j ← 1 to numberOfResources do
17             nextalloc = negsmall //  decrease resource
18             repeat
19                 apply nextalloc to resource j of VMi
20                 sleep(30sec)
21                 based on utility change apply fuzzy rules
                   and get nextalloc
22             until utility change zero or negative ;
23             apply nextalloc to resource j of VMi
24             sleep(30sec)
25         end
26     end
27     synchronization barrier() // only in multi-agent
28     send() to coordinator resource consumption and request
29     receive() from coordinator maximum resources allowed
30 end
```

VI. MULTI-AGENT FUZZY CONTROL

Since an increased number of VMs leads to an increased number of iterations of the for-loops in lines 3 and 15, the runtime of Algorithm 2 for global utility optimization increases correspondingly. Thus, we have developed a multi-agent fuzzy controller as shown in Fig. 5. Each agent is

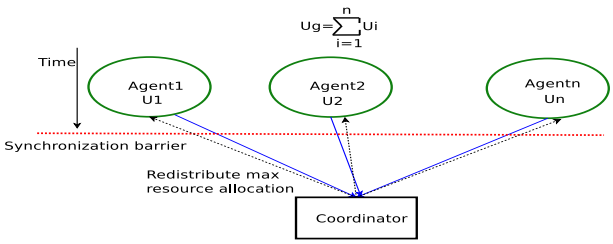


Figure 5. Multi-agent fuzzy controller architecture

responsible for the resource allocation of a single VM by optimizing its own local utility function in parallel with other agents. Since the goal is to optimize the global utility function given by (1), this function should be divided into local utility functions in such a way that optimizing each of them separately would lead to global utility optimization. The global utility function in (1) permits us to express it as the sum of local utility functions:

$$\begin{aligned} U &= \delta(\alpha\frac{V_1+V_2+..}{n} - \beta\frac{cpu_1+cpu_2..+mem_1+mem_2..}{2}) \\ &= \delta((\alpha\frac{V_1}{n} - \beta\frac{cpu_1+mem_1}{2}) + (\alpha\frac{V_2}{n} - \beta\frac{cpu_2+mem_2}{2}) \\ &... + (\alpha\frac{V_n}{n} - \beta\frac{cpu_n+mem_n}{2})) \\ &= \delta(U_1+U_2+...+U_n) \end{aligned} \tag{4}$$

where U_1,U_2,..U_n represent local utility functions that are assigned to each agent. A local utility function U_i depends only on resource costs ($\frac{cpu_i+mem_i}{2}$) and application performance V_i of the corresponding VM_i, meaning that it can be optimized separately by each agent. Since the global utility is the sum of local utilities, optimizing each of them separately optimizes the global utility.

Resources allocated to VMs in the system should not exceed the total capacity of the resources, but since the agents operate independently, they can allocate resources in such a way that this condition is violated. To solve this problem, we introduce a coordinator that periodically calculates and distributes to all agents the maximum amount of resources that can be allocated by them, obeying to the total capacity condition.

In the beginning, the coordinator assigns the same amount of maximum resources to all agents. At the time of calculation, it gets the resource consumption and the maximum resource allowed for each agent. From the difference between the two, it calculates the amount of resources that is not used by each agent. Summing up all free resources of all agents it obtains the total amount of free resources that can be redistributed to them. To do this, the total amount of free resources is divided by the number of agents, and the result is added to the resource consumption of each agent to get the maximum amount of resources allowed for each of them. This method is applied for each resource separately.

The maximum amount of resources is redistributed after each agent has passed two rounds of increasing and decreasing the resource allocations (see Algorithm 2). This scheme is fair by giving each agent the same number of rounds for optimizing its local utility functions before getting its allowed maximum amount of resources. Since agents can finish their two rounds with different speeds, they are synchronized with each other through a synchronization barrier primitive at the moment of requesting the maximum amount of resources.

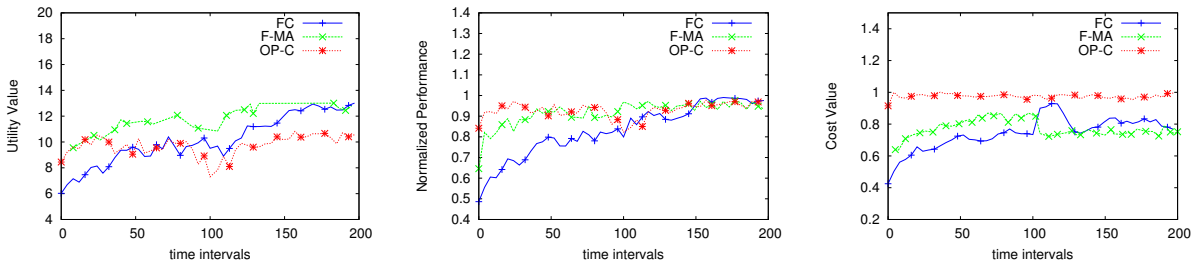
The algorithm followed by each agent is shown in Algorithm 2. In this case, the fuzzy rules are applied for allocating resources to a single VM, so $numberOfVMs = 1$. After two rounds of increasing and decreasing resource allocations, the agent is synchronized with other agents through the barrier primitive. After synchronization, it sends the actual resource consumption and the maximum resource allocation request to the coordinator. After receiving from the coordinator the maximum amount of allowed resources, it continues with the next rounds of resource allocations.

VII. Experimental Results

We have implemented the resource allocation approach in the C++ programming language. The Xen hypervisor is used as the virtualization technology. The performance sensor runs as a daemon inside each VM and is attached to an application through a shared memory mechanism of Linux to gather performance metrics using the heartbeat API framework [6]. The main sensor that is part of the resource manager requests performance metrics through the IP socket interface from all application sensors in each control interval. The actuator sets the CPU and memory allocation using the xm sched-credit and xm mem-set commands of the Xen hypervisor. We implemented the multi-agent version of the fuzzy controller as a multi-threaded program. We have set the value of α to 2 and the value of β to 1 in (1) to give more priority to performance than to resource costs.

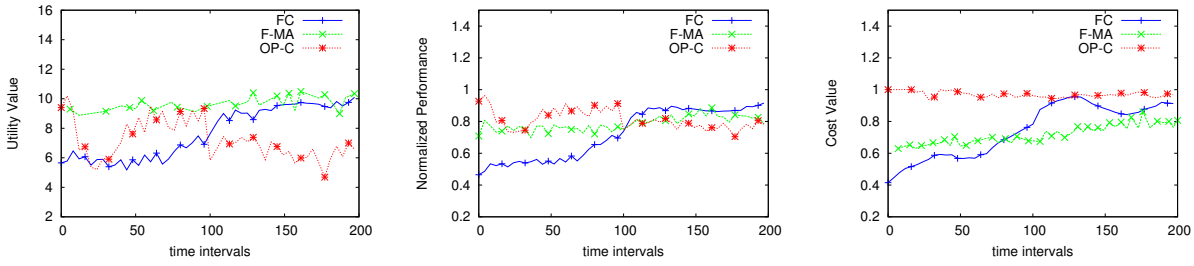
We have set up a physical machine as a testbed to run VMs to be managed. It has two dual-core AMD Opteron 2.4 GHz processors and 8 GB of RAM. It runs Xen 4.1 and the Ubuntu 12.04 operating system in the Dom0 and DomU VMs. For our evaluation, we used the following benchmarks: a) *cpu_bench* is a CPU intensive benchmark created by ourselves that performs a set of mathematical calculations, such as estimating the factorial of a number; b) *filebench* [1] is a file system benchmark that allows us to generate a variety of workloads emulating a number of applications such as web, file, and database servers.

We performed experiments with four VMs and eight VMs, where in each case half of them run filebench with the web server profile and the other half run cpu_bench. We pin four VMs in two CPU cores and eight VMs to three CPU cores and give each of them one VCPU while we pin Dom0 to another core. We give 1600 MB of memory



(a) Utility (b) Average performance (c) Average resource cost

Figure 6. Utility, average performance and average resource cost for 4 VMs



(a) Utility (b) Average performance (c) Average resource cost

Figure 7. Utility, average performance and average resource cost for 8 VMs

to four VMs, while we give 3200 MB to eight VMs. We set a minimum of resource allocation of 20% to each VM for not letting them starve for resources. We used the following resource granularities: *(neg/pos)small*=10%, *(neg/pos)large*=20% for the CPU and *(neg/pos)small*=80 MB, *(neg/pos)large*=160 MB for the memory. We ran the experiment for 200 intervals, and in interval 100 we initiated a workload change. We changed the performance levels of two filebench VMs and two cpu_bench VMs to emulate a workload change. The experiment was run for each approach five times and the average results are shown. We use the following abbreviations: Fuzzy Control - FC, Fuzzy Control Multi-Agent - F-MA, and Adaptive Optimal Control - OP-C.

In Fig. 6(a) and 7(a), the average utility over time for the three approaches is shown for 4 and 8 VMs. We see that F-MA has better average utility value during the whole experimental interval compared to the other approaches for both cases. FC has more difficulty to reach the optimal utility value with increasing the number of VMs from 4 to 8, as indicated by a larger difference of the utility value to F-MA for 8 VMs. This is because the number of resource allocations steps to reach the optimal utility increases with the number of VMs, which is not the case for F-MA. We also see that FC reaches the same final maximal utility as F-MA after the workload change in the interval from 100 to 200. This is because the resource allocation in time interval 100 changes in a few steps to obtain the resource allocation with the optimal utility in time interval 200, and thus this can be achieved faster by FC.

To understand whether the utility difference of F-MA to the other approaches is statistically significant, we performed paired t-tests over all intervals between F-MA and each of the other approaches with a 95% confidence interval. The utility value difference between F-MA and FC is statistically significant over only 37% of the intervals for the 4 VM case and 50% for the 8 VM case. This is because the advantage of F-MA can appear only with an increased number of VMs. The utility value difference between F-MA and OP-C is statistically significant over 72% of the intervals for 4 VMs and 67% for 8 VMs. The average performance over all VMs was calculated for each interval, and the average of this value over five runs for each approach is shown in Fig. 6(b) and 7(b). We see that FC has difficulties to keep the performance to desired levels especially until interval 100 with a statistically significant difference with other approaches, while F-MA is able to keep the performance near the desired level over all intervals together with OP-C.

Fig. 6(c) and 7(c) show the average resource cost over all intervals as calculated by (3). We see that FC and F-MA achieve comparable resource costs, sometimes F-MA achieves lower cost as shown in the second phase for 8 VMs. OP-C achieves higher resource costs than the other approaches, since OP-C does not take into account resource costs through optimization of the utility value, but focuses

only on performance. F-MA achieves almost the same resource costs as FC but with higher average performance resulting in higher utility values, as shown in Fig. 6(a) and 7(a). F-MA achieves the right amount of resource costs needed to keep the performance to high levels by finding the best performance-cost trade-off.

VIII. Conclusion

We have presented an approach to address the performance-cost tradeoff for VM resource allocation in cloud computing. It is based on expressing the two conflicting goals in a utility function and using fuzzy control for optimizing it. To support an increased number of VMs, we have developed a multi-agent fuzzy control approach.

Experiments comparing the multi-agent approach with centralized fuzzy control and a state-of-the-art adaptive optimal control approach show the effectiveness of the proposed multi-agent fuzzy controller. It performs better than the other approaches, especially with an increased number of VMs.

In the future, we plan to include power consumption and disk I/O bandwidth in the utility function. Furthermore, we will evaluate the approach in a complex environment with multiple physical machines and multi-tier applications.

IX. Acknowledgment

This work is supported by the German Ministry of Education and Research (BMBF) and the Albanian Government.

References

[1] Filebench: file system and storage benchmark. http://sourceforge.net/apps/mediawiki/filebench/index.php.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles*, pages 164–177. ACM Press, 2003.

[3] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proc. 2nd International Conference on Automatic Computing*, pages 229–240. IEEE Press, 2005.

[4] Y. Diao, J. L. Hellerstein, and S. Parekh. Using fuzzy control to maximize profits in service level management. *IBM Syst. J.*, 41(3):403–420, 2002.

[5] R. P. Doyle. Model-based resource provisioning in a web service utility. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, pages 5–5. USENIX Assoc., 2003.

[6] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proc. International Conference on Autonomic Computing*, pages 79–88. ACM Press, 2010.

[7] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta. Modeling virtualized applications using machine learning techniques. In *Proc. 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, pages 3–14. ACM Press, 2012.

[8] E. Loureiro, P. Nixon, and S. Dobson. Decentralized utility maximization for adaptive management of shared resource pools. In *Proc. International Conference on Intelligent Networking and Collaborative Systems*, pages 127–134. IEEE press, 2009.

[9] K. Magnus, Z. Xiaoyun, and K. Christos. An adaptive optimal controller for non-intrusive performance differentiation in computing services. In *Proc. IEEE Conference on Control and Automation*, pages 709–714. IEEE Press, 2005.

[10] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[11] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. 4th ACM European Conference on Computer Systems*, pages 13–26. ACM Press, 2009.

[12] K. M. Passino and S. Yurkovich. *Fuzzy Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.

[13] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin. Vconf: A reinforcement learning approach to virtual machines auto-configuration. In *Proc. 6th International Conference on Autonomic computing*, pages 137–146. ACM Press, 2009.

[14] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu. Dynaqos: model-free self-tuning fuzzy control of virtualized resources for qos provisioning. In *Proc. 19th International Workshop on Quality of Service*, pages 1–9. IEEE Press, 2011.

[15] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. A multi-agent systems approach to autonomic computing. In *Proc. 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 464–471. IEEE Press, 2004.

[16] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[17] VMWare Inc. VMWare Homepage. http://www.vmware.com/, 2011.

[18] Z. Wang, X. Zhu, S. Singhal, and H. Packard. Utilization and slo-based control for dynamic sizing of resource partitions. In *Proc. 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 24–26. Springer-Verlag, 2005.

[19] J. Wildstrom, P. Stone, and E. Witchel. Carve: A cognitive agent for resource value estimation. In *Proc. 5th IEEE International Conference on Autonomic Computing*, pages 182–191. IEEE Press, 2008.

[20] J. Wildstrom, P. Stone, E. Witchel, and M. Dahlin. Machine learning for on-line hardware reconfiguration. In *Proc. 20th International Joint Conference on Artifical Intelligence*, pages 1113–1118. Morgan Kaufmann Publishers Inc.