# Simulating Autonomic SLA Enactment in Clouds using Case Based Reasoning

Michael Maurer[1], Ivona Brandic[1], and Rizos Sakellariou[2]

[1] Vienna University of Technology, Distributed Systems Group, Argentinierstraße 8, 1040 Vienna, Austria, {maurer,ivona}@infosys.tuwien.ac.at
[2] University of Manchester, School of Computer Science, U.K., rizos@cs.man.ac.uk

**Abstract.** With the emergence of Cloud Computing resources of physical machines have to be allocated to virtual machines (VMs) in an on-demand way. However, the efficient allocation of resources like memory, storage or bandwidth to a VM is not a trivial task. On the one hand, the Service Level Agreement (SLA) that defines QoS goals for arbitrary parameters between the Cloud provider and the customer should not be violated. On the other hand, the Cloud providers aim to maximize their profit, where optimizing resource usage is an important part. In this paper we develop a simulation engine that mimics the control cycle of an autonomic manager to evaluate different knowledge management techniques (KM) feasible for efficient resource management and SLA attainment. We especially focus on the use of Case Based Reasoning (CBR) for KM and decision-making. We discuss its suitability for efficiently governing on-demand resource allocation in Cloud infrastructures by evaluating it with the simulation engine.

**Keywords:** Cloud Computing, Autonomic Computing, Service Level Agreement, Case Based Reasoning, Knowledge Management, Resource Management.

## 1 Introduction

The emergence of Cloud Computing – a computing paradigm that provides computing power as a utility – raises the question of dynamically allocating resources in an on-demand way. The resources that Cloud Computing providers should allocate for the customers' applications can be inferred from so-called *Service Level Agreements* (SLAs). SLAs contain *Service Level Objectives* (SLOs) that represent Quality of Service (QoS) goals, e.g., storage $\geq 1000GB$, bandwidth $\geq 10Mbit/s$ or response time $\leq 2s$, and penalties that have to be paid to the customer if these goals are violated. Consequently, on the one hand Cloud providers face the question of allocating enough resources for every application. On the other hand, however, they have to use resources as efficiently as possible: one should only allocate what is really needed.

This work is embedded in the *Foundations of Self-governing ICT infrastructure* (FoSII) project [3]. The *FoSII* project aims at developing an infrastructure

for autonomic SLA management and enforcement. Besides the already implemented *LoM2HiS* framework [12] that takes care of monitoring the state of the Cloud infrastructure and its applications, the knowledge management (KM) system presented in this paper represents another building block of the FoSII infrastructure. [11] proposes an approach to manage Cloud infrastructures by means of Autonomic Computing, which in a control loop monitors (M) Cloud parameters, analyzes (A) them, plans (P) actions and executes (E) them; the full cycle is known as MAPE [15]. According to [14] a MAPE-K loop stores knowledge (K) required for decision-making in a knowledge base (KB) that is accessed by the individual phases. This paper addresses the research question of finding a suitable KM system (i.e., a technique of how stored information should be used) and determining how it interacts with the other phases for dynamically and efficiently allocating resources.

In [19] we have argued for the use of *Case Based Reasoning* (CBR) as KM technique, because it offers a natural translation of Cloud status information into formal knowledge representation and an easy integration with the MAPE phases. Moreover, it promises to be scalable (as opposed to e.g., Situation Calculus) and easily configurable (as opposed to rule-based systems). Related work has not observed the usage of CBR nor has it evaluated different KM techniques in Cloud environments. Yet, evaluating the KM system on a real environment is not a trivial task, because Cloud infrastructures usually are huge data centers consisting of hundreds of PMs and even more VMs. Thus, a first step is to simulate the impact of autonomic management decisions on the Cloud infrastructure to determine the performance of the KM decisions.

The main contribution of this paper is the formulation of a CBR-based approach for the decision making in the MAPE cycle (in particular the analysis and planning phases) of an autonomic SLA enactment environment in Clouds. Furthermore, we design, implement and evaluate a generic simulation engine for the evaluation of CBR. With proper interfaces the engine can be used for the evaluation of other decision making techniques. Finally, we carry out an initial evaluation of the approach to assess the suitability of CBR for resource-efficient SLA management.

The remainder of this paper is organized as follows: Section 2 gives a brief overview of related work. Section 3 describes the Cloud Infrastructure we are simulating and explains the autonomic management system of FosII. While Section 4 explains the idea of the simulation engine, Section 5 provides a detailed view of its implementation and details of the adaptation of CBR. Finally, we present the evaluation of CBR as KM technique by the simulation engine in Section 6 and conclude in Section 7.

## 2 Related Work

Firstly, there has been some considerable work on optimizing resource usage while keeping QoS goals. These papers, however, concentrate on clusters, which usually lack the ability to provision computing power on demand and only deal

with heterogenous resources [17], or only deal with one or two specific SLA parameters. Petrucci [21] or Bichler [10] investigate one general resource constraint and Khanna [7] only focuses on response time and throughput. A quite similar approach to our concept is provided by the Sandpiper framework [22], which offers black-box and gray-box resource management for VMs. Contrary to our approach, though, it plans reactions just after violations have occurred. Additionally, none of the presented papers uses a KB for recording past action and learning.

Secondly, there has been work on KM of SLAs, especially rule-based systems. Paschke [20] et al. look into a rule based approach in combination with the logical formalism ContractLog. It specifies rules to trigger after a violation has occurred, but it does not deal with avoidance of SLA violations. Others inspected the use of ontologies as KBs only at a conceptual level. [18] viewed the system in four layers (i.e., business, system, network and device) and broke down the SLA into relevant information for each layer, which had the responsibility of allocating required resources. Again, no details on how to achieve this have been given. Bahati et al. [9] also use policies, i.e., rules, to achieve autonomic management. As in the other papers, this work deals with only one SLA parameter and a quite limited set of actions, and with violations and not with the avoidance thereof. Our KM system allows to choose any arbitrary number of parameters that can be adjusted on a VM.

Thirdly, compared to other SLA management projects like SLA@SOI [6], the FoSII project in general is more specific on Cloud Computing aspects like deployment, monitoring of resources and their translation into high level SLAs instead of just working on high-level SLAs in general service-oriented architectures.

## 3 FoSII Overview

In this section we describe the basic design of the Cloud infrastructure being simulated and the autonomic management cycle used in the FoSII project.

### 3.1 FoSII's Autonomic Cycle and Monitoring

As shown in Figure 1 the FoSII infrastructure is used to manage the whole lifecycle of self-adaptable Cloud services [11]. The management is governed by a *MAPE* cycle, whose components will be explained in this subsection. As part of the Monitor phase of the MAPE cycle, the host monitor sensors continuously monitor the infrastructure resource metrics and provide the autonomic manager with the current resource status. The run-time monitor sensors sense SLA violation threats based on predefined threat thresholds. *Threat thresholds* (TTs) as explained in [5] are more restrictive thresholds than the SLO values. The generation of TTs is far from trivial and should be retrieved and updated by the KM system, and only at the beginning be configured manually. Violation threats are then forwarded to the KB together with monitoring information that represents a current snapshot of the system.

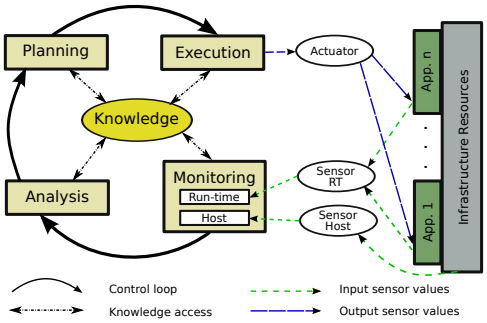


**Fig. 1.** FoSII infrastructure

Since monitoring VMs retrieves values like `free_disk` or `packets_sent`, but we are more interested in SLA parameters like storage or bandwidth, there has to be a mapping between these low level metrics and high level SLAs. This is achieved by the already mentioned highly scalable framework *LoM2HiS* [12].

After an SLA violation threat has been received by the KM during the *Analysis* phase, it has to decide which reactive action has to be taken in order to prevent the possible violation. The *Plan* phase is divided into two parts: phase *Plan I* cares about mapping actions onto PMs and managing the PMs (levels (ii) and (iii) in Subsection 3.2). Phase *Plan II* then is in charge of planning the order and timing of the actions with the additional goal of preventing oscillations, i.e., increasing and decreasing the same resources for several times. Finally the actions are executed (*Execution* phase) with the help of actuators.

### 3.2 FoSII's Infrastructure and Knowledge Management Use Case

We assume that customers deploy applications on an IaaS Cloud infrastructure. SLOs are defined within an SLA between the customer and the Cloud provider for every application. Furthermore, there is a 1:1 relationship between applications and VMs. One VM runs on exactly one PM, but one PM can host an arbitrary number of VMs with respect to supplied vs. demanded resource capacities. After allocating VMs with an initial capacity (by estimating initial resource demand) for every application, we continuously monitor actually used resources and re-allocate resources according to these measurements. Along with the process of re-allocating VM resources, the VMs have to be deployed to PMs, a VM possibly migrated from one PM to another or PMs turned on/off. Thus, one is dealing with autonomic management on three levels with respect to the following order: (i) VM resource management, (ii) VM deployment, (iii) PM management. This paper will focus on level (i). As far as level (ii) is concerned, deploying VMs on PMs with capacity constraints minimizing consumed energy by the PMs (also taking into account the costs of booting PMs and migrating VMs) can be formulated into a *Binary integer problem* (BIP), which is known to be NP-complete [16]. The proof is out of scope for this paper, but a similar ap-

proach can be seen in [21]. As realistic Cloud environments consist of many VMs and PMs, this can be considered as a real problem in practice. Consequently, heuristics have to be found to approach this problem within reasonable time. Level (iii) considers turning on/off PMs or modifying CPU frequency.

For level (i) we need to define how the measured, provided and agreed values interrelate. An example is provided in Table 1. At first, we deal with the measured value (1), which represents the amount of a specific resource that is currently used by the customer. Second, there is the amount of allocated resource (2) that can be used by the customer, i.e., that is allocated to the VM which hosts the application. Third, there is the SLO agreed in the SLA (3). A violation therefore occurs, if less is provided (2) than the customer utilizes (or wants to utilize) (1) with respect to the limits set in the SLA (3). Considering Table 1 we can see that rows 1 and 3 do not represent violations, whereas row 2 does represent an SLA violation.

| Provided (1) | Utilized (2) | Agreed (3) | Violation? |
|---|---|---|---|
| 500 GB | 400 GB | $\geq 1000GB$ | NO |
| 500 GB | 510 GB | $\geq 1000GB$ | YES |
| 1000 GB | 1010 GB | $\geq 1000GB$ | NO |

**Table 1.** Cases of (non-) SLA violations using the example of storage

## 4 Description of the simulation engine

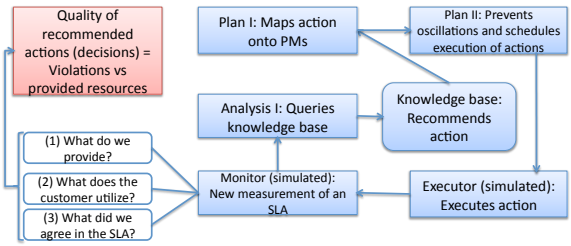


**Fig. 2.** MAPE cycle in FoSII

The goal of the simulation engine is to evaluate the quality of a KM system with respect to the number of SLA violations and the utilization of the resources. Furthermore, the simulation engine serves as an evaluation tool for any KM technique in the field of Cloud Computing, as long as it can implement the two methods of the KB management interface:

1. `public void receiveMeasurement(int slaID, String[] provided, String[] measurements, List<String> violations);` and
2. `public Action recommendAction(int slaID);`.

The parameter `slaID` describes the ID of the SLA that is tied to the specific VM, whose provided and measured values are stored in the arrays `provided` and `measurements`, respectively. The list `violations` contains all SLA parameters being violated for the current measurements. The method `receiveMeasurement` inputs new data into the KB, whereas the method `recommendAction` outputs an action specific to the current measurement of the specified SLA.

The simulation engine traverses parts of the MAPE-cycle as can be seen in Figure 2 and described in Subsection 3.1. The Monitoring and Executor part are simulated, Analysis I and the KB are implemented using CBR. Plan phases I and II are currently not considered within the simulation engine; the focus of the engine is to solve the question of how to correctly allocate resources for VMs before deploying them to PMs. Possible reactive actions have to be pre-defined and are – for the Analysis phase – tied to parameters that can directly be tuned on a VM: Increase/Decrease storage/bandwidth/memory etc. by 10%/20% etc., or do nothing.

As far as the measurements are concerned, a sensor is "installed" for every VM parameter that should be measured. The simulated Monitor component simultaneously asks the sensors for the current value of their VM parameter. Sensors can be implemented completely modularly and independently. An example sensor implementation is explained in the following: For the first measurement, it randomly draws a value from a Gaussian distribution with $\mu = \frac{SLO}{2}$ and $\sigma = \frac{SLO}{8}$, where $SLO$ describes the SLO threshold set for the current VM parameter. Then, it randomly draws a trend up or down and a duration the trend is going to last. Now, as long as the trend lasts, it increases (trend up) or decreases (trend down) the measured value for every iteration by any percentage uniformly distributed in the interval $[iBegin\%, iEnd\%]$. After the trend is over, a new trend and a new duration of the trend are selected. By doing this, we achieve a set of measurement data, where values do not behave just randomly, but follow a certain trend for a period of time that is a-priori unknown to the KM. As the intensity of the trend varies for every iteration, we deal with both, slow developments and rapid changes.

The simulation engine is iteration-based, meaning that in one iteration the MAPE cycle is traversed exactly once. In reality, one iteration could last from some minutes to about an hour depending on the speed of the measurements, the length of time the decision making takes, and the duration of the execution of the action, like for example migrating a resource intensive VM to another PM.

## 5 Implementation of CBR-based Knowledge Management

In this section we describe how CBR was implemented within the simulation engine. We first explain the idea behind CBR, then define when two cases are similar, and finally derive a utility function to estimate the "goodness" of an action in a specific situation. CBR was first built ontop of FreeCBR [4], but is now a completely independent Java framework taking into account, however, basic ideas of FreeCBR.

### 5.1 CBR Overview

Case Based Reasoning is the process of solving problems based on past experience [8]. In more detail, it tries to solve a *case* (a formatted instance of a problem) by looking for similar cases from the past and reusing the solutions of these cases to solve the current one. In general, a typical CBR cycle consists of the following phases assuming that a new case was just received:

1. Retrieve the most similar case or cases to the new one.
2. Reuse the information and knowledge in the similar case(s) to solve the problem.
3. Revise the proposed solution.
4. Retain the parts of this experience likely to be useful for future problem solving. (Store new case and found solution into KB.)

In this paragraph we formalize language elements used in the remaining paper. Each SLA has a unique identifier $id$ and a collection of SLOs. SLOs are predicates of the form

$$SLO_{id}(x_i, comp, \pi_i) \text{ with } comp \in \{<, \leq, >, \geq, =\}, \tag{1}$$

where $x_i \in P$ represents the parameter name for $i = 1, \ldots, n_{id}$, $\pi_i$ the parameter goal, and $comp$ the appropriate comparison operator. Additionally, action guarantees that state the amount of penalty that has to be paid in case of a violation can be added to SLOs, which is out of scope in this paper. Furthermore, a case $c$ is defined as

$$c = (id, m_1, p_1, m_2, p_2, \ldots, m_{n_{id}}, p_{n_{id}}), \tag{2}$$

where $id$ represents the SLA id, and $m_i$ and $p_i$ the measured (m) and provided (p) value of the SLA parameter $x_i$, respectively.

A typical use case for the evaluation might be: SLA id = 1 with $SLO_1$("Storage", $\geq$, 1000, $ag_1$) and $SLO_1$ ("Bandwidth", $\geq$, 50.0, $ag_1$), where $ag_1$ stands for the appropriate action guarantee to execute after an SLO violation. A simple case that would be received by a measurement component could therefore look like $c = (1, 500, 700, 20.0, 30.0)$. A result case $rc = (c^-, ac, c^+, utility)$ includes the initial case $c^-$, the executed action $ac$, the resulting case $c^+$ measured some time interval later (one iteration in the simulation engine) and the calculated *utility* described in Section 5.3.

### 5.2 Similarity Measurement between two Cases

In order to retrieve similar cases already stored in the database to a new one, the similarity of two cases has to be calculated. However, there are many metrics that can be considered.

The problem with Euclidean distance, for instance, is due to its symmetric nature that it cannot correctly fetch whether a case is in a state of over- or under-provisioning. Additionally, the metric has to treat parameters in a normalized way so that parameters that have a larger distance range are not

over-proportionally taken into account than parameters with a smaller difference range. (E.g., if the difference between measured and provided values of parameter $A$ always lie between 0 and 100 and of parameter $B$ between 0 and 1000, the difference between an old and a new case can only be within the same ranges, respectively. Thus, just adding the differences of the parameters would yield an unproportional impact of parameter $B$.)

This leads to the following equation whose summation part follows the principle of semantic similarity from [13]:

$$d(c^-, c^+) = \min(w_{id}, \left|id^- - id^+\right|) + \sum_{x \in P} w_x \left| \frac{(p_x^- - m_x^-) - (p_x^+ - m_x^+)}{max_x - min_x} \right|, \quad (3)$$

where $w = (w_{id}, w_{x_1}, \ldots, w_{x_n})$ is the weight vector; $w_{id}$ is the weight for non-identical SLAs; $w_x$ is the weight, and $max_x$ and $min_x$ the maximum and minimum values of differences $p_x - m_x$ for parameter $x$. As can easily be checked, this indeed is a metric also in the mathematical sense.

Furthermore, the match percentage $mp$ of two cases $c^-$ and $c^+$ is then calculated as

$$mp(c^-, c^+) = \left(1 - \frac{d(c^-, c^+)}{w_{id} + \sum_x w_x}\right) \cdot 100. \quad (4)$$

This is done because the algorithm does not only consider the case with the highest match, but also cases in a certain percentage neighborhood (initially set to 3%) of the case with the highest match. From these cases the algorithm then chooses the one with the highest utility. By calculating the match percentage, the cases are distributed on a fixed line between 0 and 100, where 100 is an identical match, whereas 0 is the complete opposite.

### 5.3 Utility Function, Utilization and Resource Allocation efficiency

To calculate the utility of an action, we have to compare the initial case $c^-$ vs. the resulting final case $c^+$. The *utility function* is composed by a violation and a utilization term weighed by the factor $0 \leq \alpha \leq 1$:

$$utility = \sum_{x \in P} violation(x) + \alpha \cdot utilization(x) \quad (5)$$

Higher values for $\alpha$ give more importance to the utilization of resources, whereas lower values to the non-violation of SLA parameters. We further note that $c(x)$ describes a case only with respect to parameter $x$. E.g., we say that a violation has occurred in $c(x)$, when in case $c$ the parameter $x$ was violated.

The function *violation* for every parameter $x$ is defined as follows:

$$violation(x) = \begin{cases} 1, & \text{No violation occurred in } c^+(x), \text{ but in } c^-(x) \\ 1/2, & \text{No violation occurred in } c^+(x) \text{ and } c^-(x) \\ -1/2 & \text{Violation occurred in } c^+(x) \text{ and } c^-(x) \\ -1 & \text{Violation occurred in } c^+(x), \text{ but not in } c^-(x) \end{cases}. \quad (6)$$

For the *utilization* function we calculate the utility from the used resources in comparison to the provided ones. We define the distance $\delta(x, y) = |x - y|$, and utilization for every parameter as

$$utilization(x) = \begin{cases} 1, & \delta(p_x^-, m_x^-) > \delta(p_x^+, u_x^+) \\ -1, & \delta(p_x^-, m_x^-) < \delta(p_x^+, u_x^+) \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

We get a utilization utility of 1 if we experience less over-provisioning of resources in the final case than in the initial one, and a utilization utility of $-1$ if we experience more over-provisioning of resources in the final case than in the initial one.

If we want to map utilization, $u$, and the number of SLA violations, $v$, into a scalar called *resource allocation efficiency* (RAE), we can achieve this by

$$\text{RAE} = \begin{cases} \frac{u}{v}, & v \neq 0 \\ u, & v = 0, \end{cases} \tag{8}$$

which reflects our evaluation goals. High utilization leads to high RAE, whereas a high number of SLA violations leads to a low RAE, even if utilization is in normal range. This can be explained by the fact that having utilization at a maximum - thus being very resource efficient in the first place - does not pay if the SLA is not fulfilled at all.

## 6 Evaluation of CBR

This section first describes which initial cases are fed into CBR and then evaluates how CBR behaved over several iterations.

As a first step, the KB has to be filled with some meaningful initial cases. This was done by choosing one representative case for each action that could be triggered. For our evaluation the SLA parameters *bandwidth* and *storage* (even though not being tied to them in any way – we could have also named them, e.g., *memory* and *CPU time*) were taken into consideration resulting into 9 possible actions "Increase/Decrease bandwidth by 10%/20%", "Increase/Decrease storage by 10%/20%", and "Do nothing".

Taking storage for example, we divide the range of distances for storage $St$ between measured and provided resources into five parts as depicted in Figure 3. We choose some reasonable threshold for every action as follows: If $p_{St} - m_{St} = -10$ then action "Increase Storage by 20%" as this already is a violation; if $p_{St}^- - m_{St} = +50$ then action "Increase Storage by 10%" as resources are already scarce but not so problematic as in the previous case; if $p_{St} - m_{St} = +100$ then action "Do nothing" as resources are neither very over- nor under-provisioned; if $p_{St} - m_{St} = +200$ then action "Decrease Storage by 10%" as now resources are over-provisioned; and we set action "Decrease Storage by 20%" when we are over the latest threshold as then resources are extremely over-provisioned. We

choose the values for our initial cases from the center of the respective intervals. Ultimately, for the initial case for the action, e.g., "Increase Storage by 20%" we take the just mentioned value for storage and the "Do nothing" value for bandwidth. This leads to $c = (id, 0, -10, 0, 7.5)$, and because only the differences between the values matter, it is equivalent to, e.g., $c = (id, 200, 190, 7.5, 15.0)$.

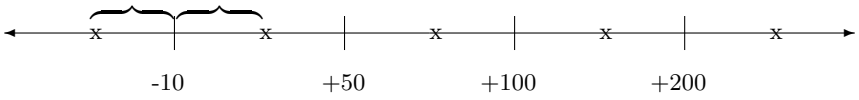


**Fig. 3.** How to choose initial cases using the example of storage

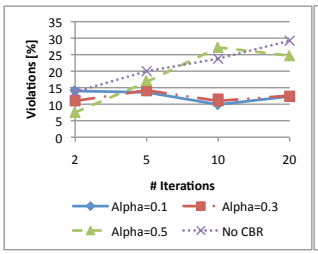


**Fig. 4.** SLA violations

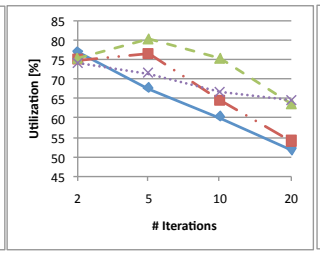


**Fig. 5.** Utilization

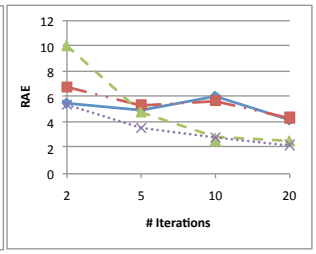


**Fig. 6.** RAE

The CBR implementation is evaluated by comparing the outcome of the simulation running with the autonomic manager and without it. At the beginning, we configure all VMs exactly equally with 80% of the storage SLO value and 2/3 of the bandwidth SLO value provided. Then, we execute 2, 5, 10 and 20 iterations with values for $\alpha$ being 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 and 0.8 (cf. Eq. 5). We omit values 0.2 and 0.4 in the evaluation because their outcomes do not differ enough from the values shown, and all values $> 0.5$, because they reveal unacceptable high SLA violation rates. After the iterations we calculate the percentage of occurred SLA violations in respect to all possible SLA violations (cf. Fig. 4), the resource utilization (cf. Fig. 5) and the RAE (cf. Fig. 6). Running without the autonomic manager means that we will leave the configuration of the VMs as they are and effect no actions due to changing demands. We set the weights $w = (1, 1, 1)$. In Figures 4 and 5 we see that the number of SLA violations and resource utilization heavily depend on the factor $\alpha$. Lower values for $\alpha$ clearly prefer to avoid SLA violations, whereas higher ones emphasize on resource utilization. We also see that with $\alpha \in \{0.1, 0.3\}$ up to more than half of the SLA violations can be avoided. However, fewer SLA violations result in lower resource utilization, as more resources have to be provided than can actually be utilized. Another point that can be observed, is that after a certain amount of iterations the quality of the recommended actions decreases. This is probably due to the fact that the initial cases get more and more blurred when more cases are stored

into CBR, as all new cases are being learned and there is no distinction being made between "interesting" and "uninteresting" cases. Nevertheless, when we relate SLA violations and resource utilization in terms of RAE, CBR methods for $\alpha \in \{0.1, 0.3\}$ are up to three times better than the default method. Summing up, the simulation shows that learning did take place and that CBR is able to recommend right actions for many cases, i.e., to correctly handle and interpret the measurement information that is based on a random distribution not known to CBR.

## 7 Conclusion

This paper presents a first attempt to create a KM system for Cloud Computing. To evaluate such systems, a KM technique-agnostic simulation engine that simulates monitoring information and executing actions on VMs in a Cloud environment has been developed. Furthermore, we implemented a CBR style knowledge base and evaluated it. The results are both encouraging and exhibit needs for further improvement. On the one hand they show that CBR reduces the number of SLA violations and increases RAE compared to a static approach. On the other hand, we can subsume two points that have to be leveraged: (i) learning techniques have to be ameliorated (step 4 in subsection 5.1) in order to maintain the high quality of the cases in the knowledge base; (ii) fine-tuning the similarity function should help to choose the most dangerous parameters with higher precision. One of the limitations of CBR is that for every parameter, the parameter space has to be divided into different areas corresponding to specific actions as in Fig. 3. This, however, could be also used for a rule-based approach, where these areas are specified with thresholds that can be learned using the utility function defined in Section 5.3. In CBR, the learning of these spaces inherently and implicitly takes place, but it would be interesting for a rule-based approach to compare whether an explicit learning or definition of the thresholds could bring a benefit to the RAE of the system. Therefore, the evaluation of other KM techniques with this simulation engine is our ongoing work. We want to compare the performance of CBR with a rule-based approach using Drools [2] and a default logic based approach based on DLV [1] in order to determine the most appropriate knowledge management method for Cloud computing.

*Acknowledgments* The work described in this paper is supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-Governing ICT Infrastructures (FoSII) and by COST-Action IC0804 on energy efficiency in large scale distributed systems.

## References

1. (DLV) - The DLV Project - A Disjunctive Datalog System, http://www.dbai.tuwien.ac.at/proj/dlv/
2. Drools, www.drools.org

3. (FOSII) - Foundations of Self-governing ICT Infrastructures, http://www.infosys.tuwien.ac.at/linksites/fosii
4. FreeCBR, http://freecbr.sourceforge.net/
5. IT-Tude: SLA monitoring and evaluation, http://www.it-tude.com/sla-monitoring-evaluation.html
6. SLA@SOI, http://sla-at-soi.eu/
7. Application Performance Management in Virtualized Server Environments (2006), http://dx.doi.org/10.1109/NOMS.2006.1687567
8. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches (1994)
9. Bahati, R.M., Bauer, M.A.: Adapting to run-time changes in policies driving autonomic management. In: ICAS '08: Proceedings of the 4th Int. Conf. on Autonomic and Autonomous Systems. IEEE Computer Society, Washington, DC, USA (2008)
10. Bichler, M., Setzer, T., Speitkamp, B.: Capacity Planning for Virtualized Servers. Presented at Workshop on Information Technologies and Systems (WITS), Milwaukee, Wisconsin, USA, 2006 (2006)
11. Brandic, I.: Towards self-manageable cloud services. In: Ahamed, S.I., et al. (eds.) COMPSAC (2). pp. 128–133. IEEE Computer Society (2009)
12. Emeakaroha, V.C., I.Brandic, Maurer, M., Dustdar, S.: Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In: The 2010 High Performance Computing and Simulation Conference in conjunction with IWCMC 2010. Caen, France (2010)
13. Hefke, M.: A framework for the successful introduction of KM using CBR and semantic web technologies. Journal of Universal Computer Science 10(6) (2004)
14. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. 40(3), 1–28 (2008)
15. Jacob, B., Lanyon-Hogg, R., Nadgir, D.K., Yassin, A.F.: A practical guide to the IBM Autonomic Computing toolkit. IBM Redbooks (2004)
16. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations: Proc. of a Symp. on the Complexity of Computer Computations. pp. 85–103. Plenum Press (1972)
17. Khargharia, B., Hariri, S., Yousif, M.S.: Autonomic power and performance management for computing systems. Cluster Computing 11(2), 167–181 (2008)
18. Koumoutsos, G., Denazis, S., Thramboulidis, K.: SLA e-negotiations, enforcement and management in an autonomic environment. Modelling Autonomic Communications Environments pp. 120–125 (2008)
19. Maurer, M., Brandic, I., Emeakaroha, V.C., Dustdar, S.: Towards knowledge management in self-adaptable clouds. In: IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems. Miami, USA (2010)
20. Paschke, A., Bichler, M.: Knowledge representation concepts for automated SLA management. Decision Support Systems 46(1), 187–205 (2008)
21. Petrucci, V., Loques, O., Mossé, D.: A dynamic optimization model for power and performance management of virtualized clusters. In: e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. pp. 225–233. ACM, New York, NY, USA (2010)
22. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Sandpiper: Black-box and gray-box resource management for virtual machines. Computer Networks 53(17), 2923 – 2938 (2009)