Future Generation Computer Systems 29 (2013) 472–487

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Adaptive resource configuration for Cloud infrastructure management

Michael Maurer [a,*], Ivona Brandic [a], Rizos Sakellariou [b]

[a] Vienna University of Technology, Distributed Systems Group, Austria

[b] University of Manchester, School of Computer Science, UK

ARTICLE INFO

Article history:
Received 30 October 2011
Received in revised form
28 June 2012
Accepted 14 July 2012
Available online 25 July 2012

Keywords:
Cloud Computing
Autonomic Computing
Rule-based system
Case-Based Reasoning
Knowledge management
Resource management

ABSTRACT

To guarantee the vision of Cloud Computing QoS goals between the Cloud provider and the customer have to be dynamically met. This so-called Service Level Agreement (SLA) enactment should involve little human-based interaction in order to guarantee the scalability and efficient resource utilization of the system. To achieve this we start from Autonomic Computing, examine the autonomic control loop and adapt it to govern Cloud Computing infrastructures. We first hierarchically structure all possible adaptation actions into so-called escalation levels. We then focus on one of these levels by analyzing monitored data from virtual machines and making decisions on their resource configuration with the help of knowledge management (KM). The monitored data stems both from synthetically generated workload categorized in different workload volatility classes and from a real-world scenario: scientific workflow applications in bioinformatics. As KM techniques, we investigate two methods, Case-Based Reasoning and a rule-based approach. We design and implement both of them and evaluate them with the help of a simulation engine. Simulation reveals the feasibility of the CBR approach and major improvements by the rule-based approach considering SLA violations, resource utilization, the number of necessary reconfigurations and time performance for both, synthetically generated and real-world data.

1. Introduction

The vision of Cloud Computing is to provide computing power as a utility, like gas, electricity or water [1]. For the underlying infrastructure this means that it has to deal with dynamic load changes, ranging from peak performance to utilization gaps. This brings up two issues: on the one hand, the management of a Cloud Computing infrastructure has to guarantee pre-established contracts despite all the dynamism of workload changes. On the other hand it has to efficiently utilize resources and reduce resource wastage. As to the former, the pre-established contracts, so called Service Level Agreements (SLAs), contain Service Level Objectives (SLOs) that represent Quality of Service (QoS) goals, e.g., "storage should be at least 1000 GB", "bandwidth should be at least 10 Mbit/s" or "response time should be less than 2 s", and penalties that have to be paid to the customer if these goals are violated.

This work can be integrated into the Foundations of Self-governing ICT Infrastructure (FoSII) project [2], but is on its own completely self-sufficient. The FoSII project aims at developing an infrastructure for autonomic SLA management and enforcement. Besides the already implemented LoM2HiS framework [3] that takes care of monitoring the state of the Cloud infrastructure and its applications, the knowledge management (KM) system presented in this article can be viewed as another building block of the FoSII infrastructure. [4] 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 [5]. According to [6] 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.

One of the imminent problems that come up when dealing with the MAPE-K loop is to define possible actions that can be executed at the end of the loop. Due to the plethora of possible reconfiguration actions in Clouds, e.g., increasing/decreasing available memory or storage for virtual machines (VMs), choosing VMs to migrate to selected physical machines (PMs), determining PMs to power on/off, etc., it is not trivial to identify the most beneficial action in a certain situation. On the one hand it is not trivial to retrieve and store all necessary information in a Cloud infrastructure. On the other hand, and more important in our work,

* Correspondence to: Argentinierstrasse 8/184-1, 1040 Wien, Austria. Tel.: +43 1 58801 18457.

E-mail addresses: maurer@infosys.tuwien.ac.at (M. Maurer), ivona@infosys.tuwien.ac.at (I. Brandic), rizos@cs.man.ac.uk (R. Sakellariou).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.07.004

dealing with the complexity of recommending an action based on this information is, as we will see, in most cases NP-hard. To tackle this, we structure all possible actions and organize them in a hierarchical model of so called *escalation levels*.

In [7,8] we have shown that approaches using Case Based Reasoning (CBR) and rules as knowledge management techniques succeed in autonomically enacting SLAs and governing important parts of Cloud computing infrastructures. Case Based Reasoning was chosen, 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. However, we determined some drawbacks of CBR as far as its learning performance and its scalability were concerned. Therefore, we also designed and implemented a rule-based knowledge management approach. Using rules [8] we managed to improve not only SLA adherence and resource allocation efficiency as discussed in [7], but also attained an efficient use of reallocation actions and high scalability.

Yet, evaluating the KM system on a real environment is not a trivial task because of two reasons: First, 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. Consequently, we designed and implemented a simulation engine that mimics the MAPE-K cycle on large Clouds. Second, workload data for a large number of VMs has to be provided as input for the simulation. We decided to go two ways: On the one hand, we generated synthetic workload data categorized into different workload volatility classes. These workload volatility classes are determined by the speed and intensity of workload change. On the other hand, we gathered real world data from monitoring scientific workflow applications in the field of bioinformatics [9]. These workflows need a huge, yet unpredictable and varying amount of resources, and are thus – due to the needed flexibility and scalability – a perfect match for a Cloud computing application [10].

The main challenge in this work is to evaluate KM techniques for autonomic SLA enactment in Cloud computing infrastructures that fulfill the three following conflicting goals: (i) achieving low SLA violation rates; (ii) achieving high resource utilization such that the level of allocated but unused resources is as low as possible; and (iii) achieving (i) and (ii) by as few time- and energy-consuming reallocation actions as possible. We will call this problem the resource allocation problem throughout the rest of the paper.

The main contributions of this paper are:

1. Design and implementation of a generic (KM-technique agnostic) simulation engine to assess the quality of the KM and decision-making techniques.
2. Partitioning the resource allocation problem for Cloud infrastructures into several subproblems by proposing escalation levels that structure all possible reaction possibilities into different subproblems using a hierarchical model.
3. Design, Implementation and Evaluation of two KM techniques for one escalation level, i.e., VM resource configuration: CBR, and the rule-based approach.
4. Application of the rule-based approach to real-world monitoring data from scientific workflow applications in the field of bioinformatics.

The remainder of this work is divided as follows: In Section 2 we present related work. Section 3 gives some background information by explaining the MAPE-K loop and the FoSII project. In Section 4 we structure the problem into the mentioned escalation levels, and in Section 5 we describe how to use the two KM techniques (CBR and rules) to tackle the resource allocation problem for a certain escalation level. Section 6 shows the evaluation of both approaches, especially focusing on the rule-based approach. Section 7 concludes this contribution and points out future work.

2. Related work

Concerning related work, we have determined four different ways to compare our work with other achievements in this area. Whereas the first level compares other works dealing with SLA enactment and resource efficiency, the second one considers the area of knowledge management, and the third one compares commercial products to our approach. Fourthly, the FoSII project is briefly related to other projects in this field.

Firstly, there has been some considerable work on optimizing resource usage while keeping QoS goals. These papers, however, concentrate on specific subsystems of Large Scale Distributed Systems, such as [11] on the performance of memory systems, or only deal with one or two specific SLA parameters. Petrucci et al. [12] or Bichler et al. [13] investigate one general resource constraint and Khanna et al. [14] only focuses on response time and throughput. A quite similar approach to our concept is provided by the Sandpiper framework [15], which offers black-box and gray-box resource management for VMs. Contrary to our approach, though, it plans reactions just after violations have occurred. Also the VCONF model by Rao et al. [16] has similar goals as presented in Section 1, but depends on specific parameters, can only execute one action per iteration and it neglects the energy consumption of executed actions. Other papers focus on different escalation levels (as described in Section 4). [17,18] focus on VM migration and [19] on turning on and off physical machines, whereas our paper focuses on VM re-configuration. Additionally, none of the presented papers uses a KB for recording past action and learning. Hoyer et al. [20] also undertake a speculative approach as in our work by overbooking PM resources. They assign VMs to PMs that would exceed their maximum resource capacities, because VMs hardly ever use all their assigned resources. Computing this allocation they also take into consideration workload correlation of different VMs. Borgetto et al. [21] tackle the trade-off between consolidating VMs on PMs and turning off PMs on the one hand, and attaining SLOs for CPU and memory on the other. However, the authors assume a static setting and do not consider dynamically changing workloads. So, e.g., they do not take the number of migrations into account. Stillwell et al. [22] in a similar setting define the resource allocation problem for static workloads, present the optimal solution for small instances and evaluate heuristics by simulations. Nathani et al. [23], e.g., also deal with VM placement on PMs using scheduling techniques. [24] react to changing workload demands by starting new VM instances; taking into account VM startup time, they use prediction models to have VMs available already before the peak occurs. Other works such as [25] have already considered the last escalation level (see Section 4), i.e., outsourcing of applications to other Clouds. Summarizing we can say that there has been a great deal of work on the different escalation levels, whereas VM configuration has not been observed yet.

Secondly, there has been work on KM of SLAs, especially rule-based systems. Paschke and Bichler [26] 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. [27] 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 and Bauer [28] also use policies, i.e., rules, to achieve autonomic management. They provide a system architecture including a KB and a learning component, and divide all possible states of the system into so called regions, which they assign a certain benefit for being in this region. A bad region would be, e.g., response time > 500 (too slow), fair region response time $<$ 100 (too fast, consuming unnecessary resources) and a good region $100 \leq$ response time ≤ 500. The actions are not structured, but are mixed together into a single rule, which makes the rules very hard to manage and to determine a salience concept behind them. However, we share the idea of defining "over-utilized", "neutral" and "under-utilized" regions. Our KM system allows us to choose any arbitrary number of resource parameters that can be adjusted on a VM. Moreover, our paper provides a more wholesome approach than related work and integrates the different action levels that work has been carried out on.

Thirdly, commercial Cloud IaaS platforms such as Amazon EC2 [29], Rackspace [30] or RightScale [31] have a very limited choice of preconfigured and static VM resource provisioning types. Amazon EC2 only offers VM instance types such as small, medium or large with predefined storage, computing units, and memory without the possibility of reconfiguring or fine-tuning them beforehand, not to mention during runtime. Rackspace only offers storage on the IaaS level, and RightScale focuses more on integrating different IaaS platforms such as Amazon EC2 or Rackspace into a holistic view.

Fourthly, compared to other SLA management projects like SLA@SOI [32], 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. Background

In this section we describe how the KM approach can be integrated within a more holistic Cloud management project that, e.g., also consists of a monitoring component. Yet, the KM approach does not depend on the specific used monitoring framework, as long as it correctly measures the current values of the parameters specified in the SLA.

In this case, the FoSII project will serve as a running example. We will describe how the KM approach relates to other components of the FoSII project. Generally, the project distinguishes between *system set-up* and *run time*. During system set-up, applications, their corresponding SLAs and used infrastructure are tailored and adapted. Once the application is deployed, we consider *monitoring*, *knowledge management* and *execution* phases during run time. In this section, in particular, we focus on the adaptation, monitoring, and knowledge management phases, as shown in Fig. 1. Thus, the MAPE-K loop is extended to the A-MAPE-K loop, where the additional *A* stands for the *adaptation* phase during system set-up. This adaptation phase, however, should not be confused with later adaptation and re-configuration of resources during system run time. Quite evidently, we especially focus on the *knowledge management* phase in this paper. The three mentioned phases are described as follows:

Adaptation As shown in Fig. 1, part 1, the *adaptation* phase comprises all steps necessary to be done before successful deployment and start of the application. This includes SLA contract establishment and tailoring of the monitoring systems for the particular application. We assume that Cloud providers register their resources to particular databases containing public SLA templates. Thereafter, Cloud users can look up resources that they want to use for the deployment of their applications. Similar to the providers, Cloud users also have an SLA template utilized for their private business processes. We assume that the private SLA template cannot be changed, since it could also be part of some other local business processes and has usually to comply with different legal and security guidelines. If matching SLA templates are found, an SLA contract can be negotiated and established and the application can be deployed.

Thus, during this phase it has to be ensured that private templates of the provider and consumers match publicly available templates. However, public and private templates may differ. A typical mismatch between templates would be between different *measurement units* of attributes, as for example for the SLO clock speed or *missing attributes*. Therefore, a mechanism is required for the automatic adaptation between different templates without changing the templates themselves. A possible solution for this is the so called *SLA mapping* approach presented in [33]. This approach can include handling of missing SLA parameters, inconsistencies between attributes and translation between different attributes. More complex adaptations would include automatic service aggregation, including third party services, if, for example, the *clock speed* attribute is completely missing in the public template, but required in the private template. A third party provider (e.g., a computer hardware reseller) could be integrated to deliver information about the *clock speed* attribute. Detailed information on the adaptation phase including the SLA mapping approach are found in [33,34].

Monitoring Current monitoring systems (e.g., ganglia [35]) facilitate monitoring only of low-level systems resources, such as `free_disk` or `packets_sent`, but SLA parameters typically are, e.g., *storage* and *outgoing bandwidth*. Thus, SLA parameters required by an application usually differ from the parameters measured by the monitoring tools. To achieve a mapping from the low-level metrics to the high-level SLA parameters, the monitoring phase should comprise two core components, namely the *host monitor* and the *run-time monitor* (see Fig. 1, part 2). The former is responsible for monitoring low-level resource metrics, whereas the latter is responsible for metric mapping, and consequently for the monitoring of SLAs and informing the KM phase about SLA violations. This monitoring framework has proven to be highly scalable and is presented in more detail in [3].

Knowledge Management Since the analysis, plan and KB parts are highly interweaved with each other, we call the ensemble of these phases the *Knowledge Management Phase* (see Fig. 1, part 3). The knowledge management component receives current information about SLA parameters of each running application from the runtime monitor of the monitoring component. Depending on the KM technique in use, the KM phase analyzes this data to determine critical situations, where either SLA parameters are about to be violated or too many resources are wasted. The analysis component receives the monitoring data, stores it in the KB and queries it to recommend an action to be executed. The plan phase maps these actions onto PMs or plans outsourcing them to other Cloud providers. Finally, the actions are executed (Execution phase) with the help of actuators. Additionally, the KB does not only enable decision making out of current data, i.e., suggesting actions to be executed, but also improving the quality of decisions by keeping track of the success or failure of previous decisions, i.e., learning.

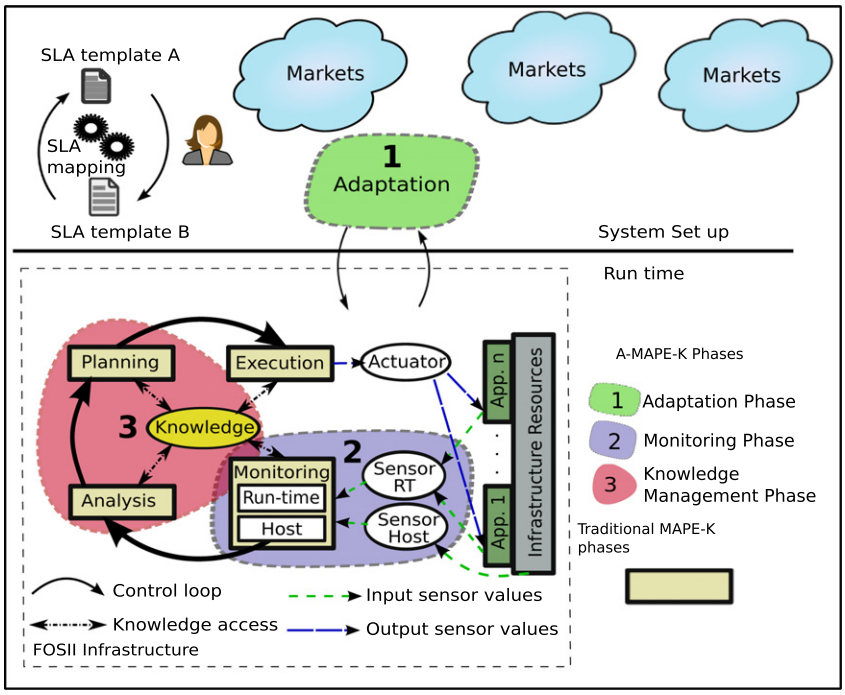


Fig. 1. FoSII infrastructure.

4. Structuring the problem: escalation levels

This section presents a methodology of dividing the resource allocation problem into smaller subproblems using a hierarchical approach. It demonstrates which actions can be executed in what level to achieve SLA adherence and efficient resource allocation for Cloud infrastructures.

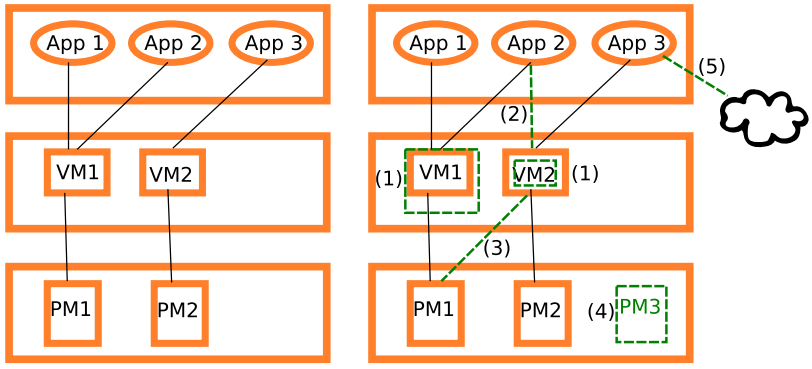
In general, we can think of the following reallocation actions:

1. for individual applications:
   (a) Increase incoming bandwidth share by *x*%.
   (b) Decrease incoming bandwidth share by *x*%.
   (c) Increase outgoing bandwidth share by *x*%.
   (d) Decrease outgoing bandwidth share by *x*%.
   (e) Increase memory by *x*%.
   (f) Decrease memory by *x*%.
   (g) Add allocated storage by *x*%.
   (h) Remove allocated storage by *x*%.
   (i) Increase CPU share by *x*%.
   (j) Decrease CPU share by *x*%.
   (k) Outsource (move application) to other Cloud.
   (l) Insource (accept application) from other Cloud.
   (m) Migrate application to different VM.
2. for VMs:
   (a) Increase incoming bandwidth share by *x*%.
   (b) Decrease incoming bandwidth share by *x*%.
   (c) Increase outgoing bandwidth share by *x*%.
   (d) Decrease outgoing bandwidth share by *x*%.
   (e) Increase memory by *x*%.
   (f) Decrease memory by *x*%.
   (g) Add allocated storage by *x*%.
   (h) Remove allocated storage by *x*%.
   (i) Increase CPU share by *x*%.
   (j) Decrease CPU share by *x*%.
   (k) Outsource (move VM) to other Cloud.
   (l) Insource (accept VM) from other Cloud.
   (m) Migrate VM to different PM.
3. for physical machines (computing nodes):
   (a) Add *x* computing nodes.
   (b) Remove *x* computing nodes.
4. Do nothing.

For an application, under "increase incoming bandwidth share" we understand to increase the application's share of all the available incoming bandwidth of a VM, and for a VM the share relates to all the available incoming bandwidth of a PM. The idea of bandwidth sharing is a common idea in network systems as described in [36]. Similar arguments account for outgoing bandwidth or CPU share.

We then group these actions into so called escalation levels that we define in Table 1. The idea is that every problem that occurs should be solved on the lowest escalation level. Only if this is not possible, the problem is tried to be solved on the next level, and again, if this fails, on the next one, and so on. The levels are ordered in a way such that lower levels offer faster and more local solutions than higher ones. At every level it has to be decided, whether the proposed action should be executed or not, because it is important to know when to do nothing, since every reallocation action is time and energy consuming. In fact, for every level there is the possibility *not* to execute the proposed action. If the proposed action is not executed, then the decision-making process will stop and not evaluate whether the next escalation level should be considered or not.

The first escalation level ("change VM configuration") works locally on a PM and tries to change the amount of storage or memory, e.g., that is allocated to the VM from the PM resources. Then, migrating applications (escalation level 2) is more lightweight than migrating VMs and turning PMs on/off (escalation levels 3 and 4). For all three escalation levels already the whole system state has to be taken into account to find an optimal solution. The problem stemming from escalation level 3 alone can be formulated into a binary integer problem (BIP), which is known to be NP-complete [37]. The proof is out of scope for this paper, but a similar approach can be seen in [12]. The last escalation level has least locality and greatest complexity, since the capacity of other



(a) Before action execution. (b) After action execution.

Fig. 2. Actions used in five escalation levels: before and after action execution.

Table 1
Escalation levels.

1.	Change VM configuration.
2.	Migrate applications from one VM to another.
3.	Migrate one VM from one PM to another or create new VM on appropriate PM.
4.	Turn on/off PM.
5.	Outsource to other Cloud provider.

Cloud infrastructures have to be taken into account, too, and negotiations have to be started with them as well.

Also the rule-based approach benefits from this hierarchical action level model, because it provides a salience concept for contradicting rules. Without this concept it would be troublesome to determine which of the actions, e.g., "Power on additional PM with extra storage and migrate VM to this PM", "Increase storage for VM by 10%" or "Migrate application to another VM with more storage" should be executed, if a certain threshold for allocated storage has been exceeded. The proposed KM approaches will present a solution for escalation level 1. Fig. 2 visualizes the escalation levels from Table 1 before and after actions are executed. Fig. 2(a) shows applications App1 and App2 deployed on VM1 that is itself deployed on PM1, whereas App3 runs on VM2 running on PM2. Fig. 2(b) shows example actions for all five escalation levels. The legend numbers correspond to the respective numbering of the escalation levels.

- *Escalation level* 1: At first, the autonomic manager considers whether it should change VM configuration or not. Actions (1) show that the autonomic manager decided to change the VM configuration; *VM1* is being up-sized and *VM2* being down-sized.
- *Escalation level* 2: If VM reconfiguration has taken place, or if it has been recommended, but cannot be fulfilled yet, because some resource cannot be increased anymore due to the constraints of the PM hosting the VM, in level 2 the autonomic manager considers migrating the application to another larger VM that fulfills the required specifications from level 1. So if, e.g., provided storage needs to be increased from 500 to 800 GB, but only 200 GB are available on the respective VM, then the application has to be migrated to a VM that has at least the same resources as the current one plus the remaining 100 GB of storage. Action (2) shows the re-deployment of *App2* to *VM2*. Due to possible confinements of some applications to certain VMs, e.g., a user deployed several applications that need to work together on one VM, this escalation might be skipped in some scenarios. Also for Infrastructure as a Service (IaaS) providers, who directly provide the VMs without caring about the applications running on them, this escalation level is omitted.
- *Escalation level* 3: If there is no appropriate VM available in level 2, or if level 2 is skipped and VM configurations have been recommended in level 1, in level 3 the autonomic manager considers creating a new VM on an appropriate PM or migrating the VM to a PM that has enough available resources. Action (3) shows the re-deployment of *VM2* to *PM1*.
- *Escalation level* 4: Again, if there is no appropriate PM available in level 3, the autonomic manager suggests turning on a new PM (or turning it off if the last VM was emigrated from this PM) in level 4. Action (4) shows powering on a new PM (*PM3*).
- *Escalation level* 5: Finally, the last escalation level 5 tries to outsource the application to another Cloud provider as explained, e.g., in the Reservoir project [38]. Action (5) outsources *App3* to another Cloud provider.

For an IaaS provider omitting escalation level 2, the sequence of these escalation levels is quite obvious: If VM sizes are not changed in escalation level 1, there is no need to trigger escalation level 3 as VMs have not changed, and no better allocation of VMs to PMs can be found, if the previous one was already optimal. However, if VM sizes were changed, escalation level 3 can still come to the conclusion that VM migrations are unnecessary. On the other hand, if VM migrations were recommended, some PMs could be then turned off in escalation level 4. Similarly, if no migrations were triggered, thinking about turning off PMs is unnecessary, as no PMs run idle now that have not been running idle before. Finally, if all the previous actions were successfully executed without the help of another Cloud provider, there is no need to consider outsourcing applications. Only if the last possibility failed, outsourcing applications should be considered. (Other business incentives for outsourcing applications such as cheaper execution costs in other Clouds, etc., are not considered here.) For providers of other Cloud delivery models such as SaaS or PaaS, the sequence of placing application migration after VM reconfiguration is arguable; another model could also propose an inverse sequence for these two levels.

5. Implementing the knowledge management phase

In this section we present the implementation of the Knowledge Management phase using CBR and a rule-based approach.

5.1. Prerequisites

This subsection subsumes all the common assumptions for both approaches.

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.

Table 2
Cases of (non-) SLA violations using the example of storage.

Provided (1) (GB)	Utilized (2) (GB)	Agreed (3) (GB)	Violation?
500	400	1000	No
500	510	1000	Yes
1000	1010	1000	No

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.

For tackling the resource allocation for VMs, we need to define how measured, provided and agreed values interrelate, and what actually constitutes an SLA violation. An example is provided in Table 2. 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 2 we can see that rows 1 and 3 do not represent violations, whereas row 2 does represent an SLA violation. In order to save resources we envision a *speculative approach*: Can we allocate less than agreed, but still more than used in order not to violate an SLA? The most demanding questions are how much can we lower the provisioning of resource without risking an SLA violation. This heavily depends on the characteristics of the workload of an application, especially its volatility.

5.2. Case Based Reasoning

Case Based Reasoning is the process of solving problems based on past experience [39]. 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 in KB.)

To adapt CBR to our problem, three issues have to be solved. First, it has to be decided how to format an instance of the problem. Second, it has to be decided when two cases are similar. Third, good reactions have to be distinguished from bad reactions.

As to the first problem we assume that each SLA has a unique identifier *id* and a collection of SLOs. SLOs are predicates of the form

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

where $x_i \in P$ represents the parameter name for $i = 1, \ldots, n_{id}$, π_i the parameter goal, and *comp* the appropriate comparison operator. Then, a CBR 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.

To use the SLA parameters *storage* and *incoming bandwidth* for example, a typical use case looks like this: SLA $id = 1$ with SLO_1 (“Storage”, $\geq$, 1000) and SLO_1 (“Bandwidth”, $\geq$, 50.0). A corresponding case received by the measurement component is therefore written as $c = (1, 500, 700, 20.0, 30.0)$. 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, which corresponds to one iteration in the simulation engine, and the calculated *utility* described later. In order to give the KB some knowledge about what to do in specific situations, several initial cases are stored in the KB as described in [7] in more detail.

Secondly, to define similarity between two cases is not straightforward, because due to their symmetric nature Euclidean distances, e.g., do not recognize the difference between over- and under-provisioning. Following the principle of semantic similarity from [40] for the summation part this leads to the following equation

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

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 far as the third issue is concerned, every action is evaluated by its impact on violations and utilization. This way CBR is able to learn whether an action was appropriate for a specific measurement or not. The utility of an action is calculated by comparing the initial case c^- with the resulting final case c^+. The *utility function* is composed by a violation and a utilization term weighed by the factor $0 \leq \alpha \leq 1$:

$$utility = \sum_{x \in P} violation(x) + \alpha \cdot utilization(x). \tag{4}$$

Higher values for α strengthen the utilization of resources, whereas lower values 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.

We define the *violation* function for every parameter x as follows:

$$violation(x) = \begin{cases} 1, & \text{No violation occurred in } c^+(x), \text{ but in } c^-(x) \\ 1/2, & \text{No violation occurred in } c^+(x) \text{ and } c^-(x) \\ -1/2 & \text{Violation occurred in } c^+(x) \text{ and } c^-(x) \\ -1 & \text{Violation occurred in } c^+(x), \text{ but not in } c^-(x). \end{cases} \tag{5}$$

The *utilization* function is calculated by comparing the used resources to the provided ones. We define the distance $\delta(x, y) = |x - y|$, and utilization for every parameter as

$$utilization(x) = \begin{cases} 1, & \delta(p_x^-, m_x^-) > \delta(p_x^+, u_x^+) \\ -1, & \delta(p_x^-, m_x^-) < \delta(p_x^+, u_x^+) \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

A utilization utility of 1 is retrieved if less over-provisioning of resources takes place in the final case than in the initial one, and a utilization utility of -1 if more over-provisioning of resources takes place in the final case than in the initial one.

The whole CBR process works as follows: Before the first iteration, we store the mentioned initial cases consisting of an initial measurement, an action and a resulting measurement. Then, when CBR receives a new measurement, this measurement is compared to all cases in the KB. From the set of closest cases

Table 3
Resource policy modes.

Green	Plenty of resources left. Over-consumption allowed.
Green–orange	Heavy over-consumption is forbidden. All applications that consume more than τ% (threshold to be specified) of the agreed resource SLO are restrained to $\tau/2$% over-consumption
Orange	Resource is becoming scarce, but SLA demand can be fulfilled if no over-consumption takes place. Thus, over-provisioning is forbidden.
Orange–red	Over-provisioning forbidden. Initiate outsourcing of some applications.
Red	Over-provisioning forbidden. SLA resource requirements of all consumers cannot be fulfilled. If possible, a specific choice of applications is outsourced. If not enough, applications with higher reputation points or penalties are given priority over applications with lower reputation points/penalties. SLAs of latter ones are deliberately broken to ensure SLAs of former ones.

grouped by a clustering algorithm we choose the one with the highest utility and execute exactly the same action as in the chosen case. Afterwards, this action, the resulting measurement and the utility of the action is added to the initial measurement, and stored as a complete case.

5.3. Rule-based approach

For the rule-based approach we first introduce several resource policy modes to reflect the overall utilization of the system in the VM configuration rules. Dealing with SLA-bound resource management, where resource usage is paid for on a "pay-as-you-go" basis with SLOs that guarantee a minimum capacity of these resources as described above, raises the question, whether the Cloud provider should allow the consumer to use more resources than agreed. We will refer to this behavior as *over-consumption*. Since the consumer will pay for every additional resource, it should be in the Cloud provider's interest to allow over-consumption as long as this behavior does not endanger the SLAs of other consumers. Thus, Cloud providers should not allow over-consumption when the resulting penalties they have to pay are higher than the expected revenue from over-consumption. To tackle this problem, we introduce five policy modes for every resource that describe the interaction of the five escalation levels. As can be seen in Table 3 the policy modes are green, green–orange, orange, orange–red and red. They range from low utilization of the system with lots of free resources left (policy mode green) over a scarce resource situation (policy mode orange) to an extremely tight resource situation (policy mode red), where it is impossible to fulfill all SLAs to their full extent and decisions have to be made which SLAs to deliberately break and which applications to outsource.

In order to know whether a resource r is in danger of under-provisioning or already is under-provisioned, or whether it is over-provisioned, we calculate the current utilization $ut^r = \frac{\text{use}^r}{\text{pr}^r} \times 100$, where use^r and pr^r signify how much of a resource r was used and provided, respectively, and divide the percentage range into three regions using the two "threat thresholds" TT^r_{low} and TT^r_{high}:

- Region -1: Danger of under-provisioning, or under-provisioning ($> TT^r_{\text{high}}$).
- Region 0: Well provisioned ($\leq TT^r_{\text{high}}$ and $\geq TT^r_{\text{low}}$).
- Region $+1$: Over-Provisioning ($< TT^r_{\text{low}}$).

The idea of this rule-based design is that the ideal value that we call *target value* $tv(r)$ for utilization of a resource r is exactly in the center of region 0. So, if the utilization value after some measurement leaves this region by using more (Region -1) or less resources (Region $+1$), then we reset the utilization to the target value, i.e., we increase or decrease allocated resources so that the utilization is again at

$$tv(r) = \frac{TT^r_{\text{low}} + TT^r_{\text{high}}}{2}\%.$$

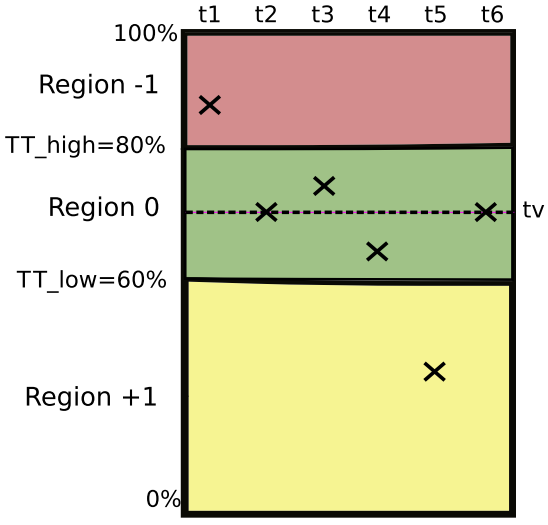


Fig. 3. Example behavior of actions at time intervals t_1–t_6.

As long as the utilization value stays in region 0, no action will be executed. E.g., for r = storage, $TT^r_{\text{low}} = 60\%$, and $TT^r_{\text{high}} = 80\%$, the target value would be $tv(r) = 70\%$. Fig. 3 shows the regions and measurements (expressed as utilization of a certain resource) at time steps $t_1, t_2, \ldots, t_6$. At t_1 the utilization of the resource is in Region -1, because it is in danger of a violation. Thus, the KB recommends to increase the resource such that at the next iteration t_2 the utilization is at the center of Region 0, which equals the target value. At time steps t_3 and t_4 utilization stays in the center region and consequently, no action is required. At t_5, the resource is under-utilized and so the KB recommends the decrease of the resource to $tv(r)$, which is attained at t_6. Additionally, if over-provisioning is allowed in the current policy mode, then the adjustment will always be executed as described regardless of what limit was agreed in the SLA. On the other hand, if over-provisioning is not allowed in the current policy mode, then the rule will allocate at most as much as agreed in the SLA (SLO^r).

The concept of a rule increasing resource r is depicted in Fig. 4. The rule executes if the current utilization ut^r and the predicted utilization $ut^r_{predicted}$ of the next iteration (cf. next paragraph) both exceed TT^r_{high} (line 2). Depending on what policy level is active the rule either sets the provided resource pr^r to the target value $tv(r)$ for policy levels green and green–orange (line 3) or to at most what was agreed in the SLA (SLO^r) plus a certain percentage ϵ to account for rounding errors when calculating the target value in policy levels orange, orange–red and red (line 5). A similar rule scheme for decreasing a resource can be seen in Fig. 5. The main difference is that it does not distinguish between policy modes and that it sets the provisioned resource to at least a minimum value $minPr^r$, which may be 0, that is needed to keep the application alive

1 **IF**
2 $ut^r > TT^r_{\text{high}}$ AND $ut^r_{predicted} > TT^r_{\text{high}}$
3 **THEN**
4 Set pr^r to $\frac{\text{use}^r}{tv(r)}$ for policy modes green, green-orange.
5 Set pr^r to $\min(\frac{\text{use}^r}{tv(r)}, SLO^r * (1 + \epsilon/100))$ for policy modes orange, orange-red, red.

Fig. 4. Rule scheme for increasing a resource.

1 **IF**
2 $ut^r < TT^r_{\text{low}}$ AND $ut^r_{predicted} < TT^r_{\text{low}}$
3 **THEN**
4 Set pr^r to $\max(\frac{\text{use}^r}{tv(r)}, minPr^r)$.

Fig. 5. Rule scheme for decreasing a resource.

(line 4). The rule is executed if the current utilization ut^r and the predicted utilization $ut^r_{predicted}$ of the next iteration both lie below TT^r_{low} (line 2).

A large enough span between the thresholds TT^r_{low} and TT^r_{high} helps to prevent oscillations of repeatedly increasing and decreasing the same resource. However, to further reduce the risk of oscillations, we suggest to calculate a prediction for the next value based on the latest measurements. Thus, an action is only invoked when the current AND the predicted measurement exceed the respective TT. So, especially when only one value exceeds the TT, no action is executed.

The rules have been implemented using the Java rule engine Drools [41]. The Drools engine sets up a knowledge session consisting of different rules and a working memory. Rules get activated when specific elements are inserted into the working memory such that the conditional "when" part evaluates to true. Activated rules are then triggered by the simulation engine. In our case, the simulation engine inserts measurements and SLAs of applications into the working memory. Different policy modes will load slightly modified rules into the Drools engine and thus achieve a high adaptability of the KM system reacting to the general performance of the Cloud infrastructure. As opposed to the CBR approach in [7], the rule-based approach is able to fire more than one action at the same iteration, which inherently increases the flexibility of the system. Without loss of generality we can assume that one application runs on one VM (several applications' SLAs can be aggregated to form one VM SLA) and we assume the more interesting case of policy modes orange, orange–red or red, where over-provisioning is not allowed.

Listing 1 shows the rule to increase parameter storage formulated in the Drools language following the pattern presented in Fig. 4. Line 1 defines the name of the rule that is split into a condition part (*when*, lines 2–12) and an execution part (*then*, lines 13–17). Line 4 tries to find the SLA of an application, and stores its id in `$slaID` and the SLA into `$slaApp`. Line 6 looks for a set of actions for this `$slaID` where no storage action has been added yet (`storage == false`) in order to avoid contradicting actions for storage for one measurement. Line 8 searches for a measurement for the appropriate VM (`vmID == $slaID`) that has been inserted into working memory that is no prediction (`$prediction == false`) and where the percentage of utilized storage exceeds TT^r_{high} (`storage_utilized > storage_HighTT`), and stores used and provided values into `$s_used` and `$s_provided`, respectively. The predicted measurement for the next iteration is handled similarly in line 10. Finally, line 12 checks whether provided storage is still below the agreed value in the SLA. This is done, because in policy modes orange to red over-consumption is prohibited. The rules for policy modes green and green–orange would omit this line. Now, if all these conditions are met, the rule gets activated. When fired, line 15 calculates the new value for pr^r as explained in Fig. 4. This line (as line 12) would also be altered for policy modes green and green–orange. Line 17 then modifies the action container `$as` and inserts the appropriate storage action with the value for provided storage to be set. Other rules follow the same pattern as described here and in Fig. 4 for rules increasing resource allocations and in Fig. 5 for rules decreasing resource allocations.

Listing 1: Rule "storage_increase"

```
1  rule "storage_increase"
2  when
3   //Remember SLA id of application
4   $SLA_app : Application($slaID : id)
5   //Look for set of actions that has no storage action yet
6   $ as: Actions(slaID == $slaID, storage == false)
7   //Look for measurement that has high utilization of
        storage
8   $m : Measurement (prediction == false, storage_utilized >
        storage_HighTT, vmID == $slaID, $s_used: storage_used
        , $s_provided: storage_provided)
9   //Look for predicted measurement that will have high
        utilization of storage
10  $m_pred : Measurement (prediction == true,
        storage_utilized > storage_HighTT, vmID == $slaID)
11  //Check whether we provide less than SLO value
12  eval($s_provided <= Double.valueOf( $SLA_app.
        getThresholdByName("storage")))
13 then
14  //Calculate tv
15  double newStorage = Math.min($s_used/((storage_HighTT+
        storage_LowTT)/2), Double.valueOf (SLA_app.
        getThresholdByName("storage"))* (1+eps/100));
16  //Add storage action to set of actions
17  modify ($ as) addAction(new StorageActionDirect(newStorage
        , "GB")), setStorage();
18 end
```

6. Evaluation and comparison

In this section we evaluate the two presented approaches with several different synthetic and real-world workload data. For this purpose, we present a KM-agnostic simulation engine that implements the autonomic control loop and simulates executed actions and evaluates their quality responding to the workload data at stake.

6.1. Simulation engine implementing the MAPE-K loop

The goal of the simulation engine is to evaluate the quality of a KM system with respect to the number of SLA violations, the utilization of the resources and the number of required reallocation actions. 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 Actions recommendAction(int slaID);`.

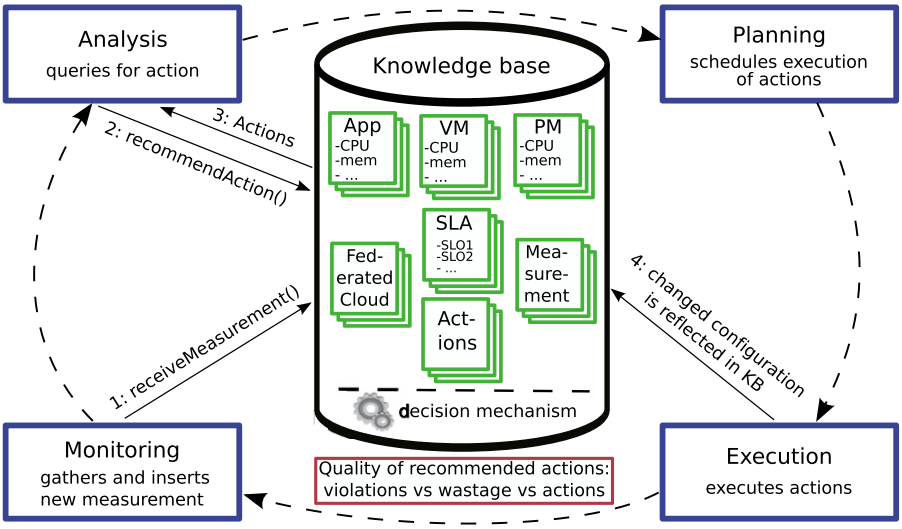


Fig. 6. Simulation engine implementing MAPE-K loop.

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 (cf. Section 5.1). 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 `recommendActions` outputs an action specific to the current measurement of the specified SLA. The simulation engine traverses all parts of the MAPE-K loop as can be seen in Fig. 6 and described in Section 3. The simulation engine is iteration based, meaning that in one iteration the MAPE-K loop 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 actions, for example migrating a resource intensive VM to another PM.) The *Monitoring* component receives monitoring information from either synthetic or real-world workload from the current iteration. It forwards the data into the *Knowledge base* (1). The Knowledge base contains representations of all important objects in the Cloud and their characteristic information. These objects are the running applications, the virtual machines, and the physical machines with the current state of their CPU power, memory, storage, etc., the corresponding SLAs with their SLOs, and information about other Clouds in the same federation. Furthermore, the KB also has representations of the inserted measurements, and the available actions to execute (these have to be pre-defined). Finally, the KB also contains a decision mechanism that interprets the state of available objects in order to recommend a reconfiguration action. This mechanism can be substituted by any KM technique; as already mentioned, we used CBR and a rule-based mechanism. The next step in the MAPE loop is the *Analysis* component, which queries the KB for actions to recommend (for a specific SLA id) (2); these actions are then returned to the analysis component (3). The *Planning* component schedules the suggested actions, and the *Execution* component executes them. The changed state configuration of the Cloud objects are automatically reflected in the KB (4). The Monitoring and the Execution components are simulated. This means that the monitoring data is not measured on a real system during the simulation, even though it handles input measured at a real system or synthetic workloads generated beforehand (see Sections 6.3 and 6.4). The Execution component updates the object representation of the manipulated objects in the KB, but obviously does not actually manipulate real-world objects. The quality of the decision making can ultimately be judged by the number of occurred SLA violations, resource wastage and the number of needed reallocation actions.

6.2. Performance indicators

The subsequent evaluations will be based on the following performance indicators: violations, utilization, actions, resource allocation efficiency (RAE), costs, and time efficiency. Whereas the first three and the last one are rather self-explanatory, costs and RAE need a little more explanation. So violations and actions measure (as a percentage) the amount of occurring violations/actions in relation to all possible violations/actions, and utilization the average utilization over all iterations (and over all SLA parameters, if they are not shown explicitly). Time efficiency measures the average time that is needed to handle one VM in one iteration. For resource allocation efficiency we want to relate violations and utilization. The basic is idea is that RAE should equal utilization ($100\% - w$, where w stands for wastage, see below) if no violations occur ($p = 0\%$, where p stands for penalty, see below), equal 0 if the violation rate is at 100%, and follow a linear decrease in between. Thus, we define

$$\text{RAE} = \frac{(100 - w)(100 - p)}{100}. \tag{7}$$

A more general approach also taking into account the cost of actions represents the definition of a generic cost function that maps SLA violations, resource wastage and the costs of executed actions into a monetary unit, which we want to call *Cloud EUR*. First, we define a penalty function $\mathfrak{p}^r(p) : [0, 100] \to \mathbb{R}^+$ that defines the relationship between the percentage of violations p (as opposed to all possible violations) and the penalty for a violation of resource r. Second, we define a function wastage $\mathfrak{w}^r(w) : [0, 100] \to \mathbb{R}^+$ that relates the percentage of unused resources w to the energy in terms of money that these resources unnecessarily consume. Third, we define a cost function $\mathfrak{a}^r(a) : [0, 100] \to \mathbb{R}^+$ from the percentage of executed actions a (as opposed to all possible actions that could have been executed) to the energy and time costs in terms of money. The total cost c is then defined as

$$c(p, w, c) = \sum_r \mathfrak{p}^r(p) + \mathfrak{w}^r(w) + \mathfrak{a}^r(a). \tag{8}$$

We assume functions $\mathfrak{p}^r$, $\mathfrak{w}^r$ and $\mathfrak{a}^r$ for this evaluation with $\mathfrak{p}^r(p) = 100p$, $\mathfrak{w}^r(w) = 5w$, and $\mathfrak{a}^r(a) = a$ for all r. The intention behind choosing these functions is (i) to impose very strict fines in order

to proclaim SLA adherence as top priority, (ii) to weigh resource wastage a little more than the cost of actions.

The cost function is currently not evaluated within the simulation engine, it is a value calculated after the simulation for comparison reasons. Thus, the recommended actions do not depend on the specific functions we assumed. However, it could be incorporated into the KB in order to adjust and learn the TTs for every resource r.

6.3. Evaluation and comparison of CBR and rules using synthetic data

To evaluate a great variety of workload data, one approach is to create them synthetically. For this, we extended the workload generator as described in [7] to allow a categorization of data volatility.

The workload generator is intended to generate very general workloads for IaaS platforms dealing with slower developments as well as rapid changes. For one parameter, the workload is generated as follows: The initial value of the workloads is randomly drawn from a Gaussian distribution with $\mu = \frac{\text{SLO}}{2}$ and $\sigma = \frac{\text{SLO}}{8}$, where SLO represents the Service Level Objective value agreed in the SLA. Then, an up- or down-trend is randomly drawn, as well as a duration of this trend between a pre-defined number of iterations (for our evaluation this interval of iterations equals $[2, 6]$), both with equal probability. For every iteration, as long as the trend lasts, the current measured value is increased or decreased (depending on the trend) by a percentage evenly drawn from the interval [*iBegin*, *iEnd*]. After the trend is over, a new trend is drawn and the iterations continue as described before.

Clearly, the values for *iBegin* and *iEnd* determine the difficulty for handling the workload. A workload that operates with low *iBegin* and *iEnd* values exhibits only very slight changes and does not, consequently, need a lot of dynamic adaptations. Large *iEnd* values, on the contrary, need the enforcement mechanisms to be very elastically tuned. For the evaluation and comparison of CBR and the rule-based approach we defined a LOW_MEDIUM workload volatility class with $iEnd = 18\%$. For the further evaluation of the rule-based approach we defined and tested LOW, MEDIUM, MEDIUM_HIGH and HIGH workload volatility classes (not shown here) with $iEnd = 10\%, 50\%, 75\%$, and 100%, respectively. As a minimum change we set $iBegin = 2\%$ for all classes.

As the crucial parameters for CBR and the rule-based approach differ, we define scenarios for both approaches separately, but still compare them to the aforementioned six performance indicators.

As resources for IaaS one can use all parameters that can be adapted on a VM. For the evaluation we chose to take the following parameters and SLOs for CBR: *storage* $\geq$ 1000 GB, *incoming bandwidth* $\geq$ 20 Mbit/s, and the following parameters and SLOs for the rule-based approach: *storage* $\geq$ 1000 GB, *incoming bandwidth* $\geq$ 20 Mbit/s, *outgoing bandwidth* $\geq$ 50 Mbit/s, *memory* $\geq$ 512 MB, and *CPU power* $\geq$ 100 MIPS (Million Instructions Per Second).

As far as CBR is concerned, its behavior differs by the α value in Eq. (4) (setting importance to avoiding violations or achieving high utilization), by the number of executed iterations, because of its inherent learning feature, and the initial cases. At the beginning, we configure all 50 VMs exactly equally with 80% of the storage SLO value and two-thirds of the bandwidth SLO value provided. Then, we execute 2, 5, 10 and 20 iterations with values for α being 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 and 0.8. 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. Setting up the initial cases 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 in nine 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 Fig. 7. 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., $c = (id, 200, 190, 7.5, 15.0)$.

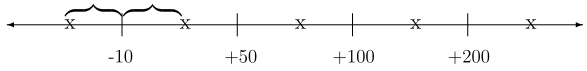


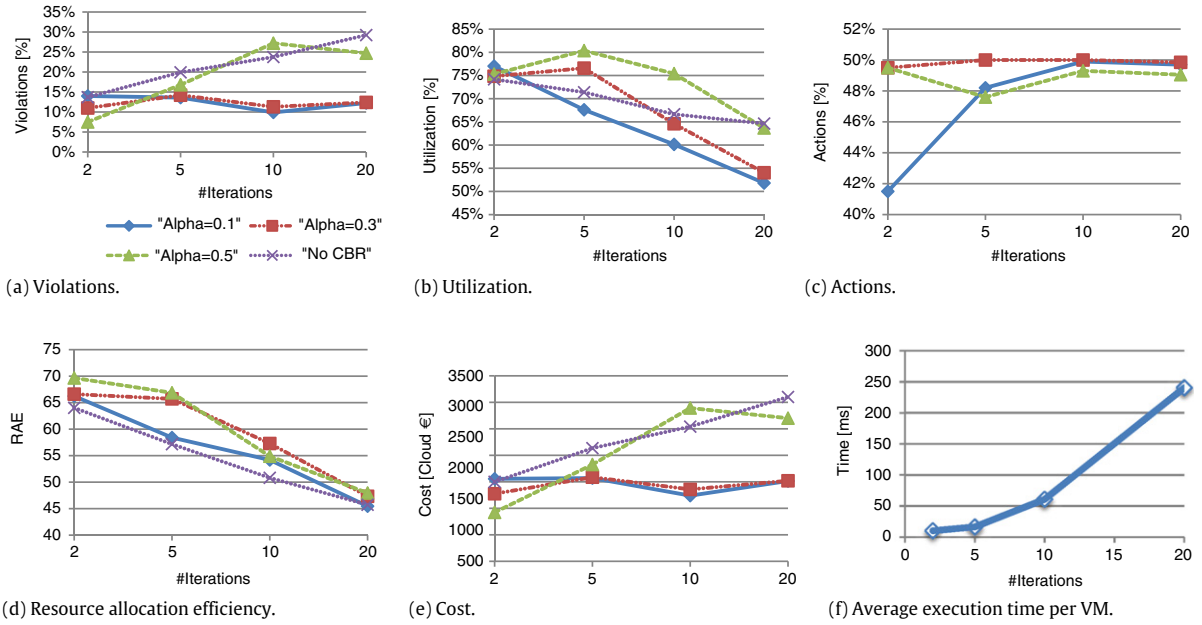
Fig. 7. Choosing initial cases for CBR using the example of storage.

Table 4
Eight simulation scenarios for TT_{low} and TT_{high}.

	Scenarios							
	1 (%)	2 (%)	3 (%)	4 (%)	5 (%)	6 (%)	7 (%)	8 (%)
TT_{low}	30	30	30	50	50	50	70	70
TT_{high}	60	75	90	60	75	90	75	90

As far as the rule-based approach is concerned, its behavior differs by the set threat thresholds. Thus, we investigate low, middle and high values for TT^r_{low} and TT^r_{high} (as defined in Section 5.3), where $TT^r_{\text{low}} \in \{30\%, 50\%, 70\%\}$ and $TT^r_{\text{high}} \in \{60\%, 75\%, 90\%\}$ for all resources stated above. We combine the TTs to form eight different scenarios as depicted in Table 4. We execute 100 iterations with 500 applications, and set the "safety slack" $\epsilon = 5\%$ (cf. Listing 1).

Fig. 8 presents the aforementioned performance indicators of CBR. The "No CBR" line means that the autonomic manager is turned off, which implies that the configuration of the VMs is left as set at the beginning, i.e., no adaptation actions due to changing demands are executed. In Fig. 8(a) we see that up to more than half of the violations can be avoided when using $\alpha \in \{0.1, 0.3\}$ instead of no autonomic management. However, fewer SLA violations result in lower resource utilization (cf. Fig. 8(b)), as more resources have to be provided than can actually be utilized. Reconfiguration actions as depicted in Fig. 8(c) lie slightly below or at 50%, except for "No CBR", of course. 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 made between "interesting" and "uninteresting" cases. Nevertheless, when we relate SLA violations and resource utilization in terms of RAE, all CBR methods are generally better than the default method, especially for $\alpha \in \{0.3, 0.5\}$ after five iterations. Yet, RAE decreases strictly monotonically for all α. Furthermore, costs – relating violations, utilization and reconfiguration actions – can also be reduced to half for $\alpha \in \{0.1, 0.3\}$. However, there is a



(a) Violations. (b) Utilization. (c) Actions.

(d) Resource allocation efficiency. (e) Cost. (f) Average execution time per VM.

Fig. 8. Evaluation of CBR with respect to SLA violations, utilization and energy efficiency.

seemingly exponential increase in the average execution time per VM (cf. Fig. 8(f)) due to higher number of cases stored in the KB.

Summing up, the simulation shows that learning did take place (and cost some time) 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.

Fig. 9 shows the same evaluation for the rule-based approach evaluating the aforementioned eight scenarios. From Fig. 9(a) we learn that in terms of SLA violations Scenario 1 achieves the best result, where only 0.0908% of all possible violations occur, and Scenario 8 yields the worst result, with a still very low violation rate of 1.2040%. In general, the higher the values are for TT_{high}, the worse is the outcome. The best result achieved with CBR was at 7.5%. Thus, the rule-based approach achieves an up to 82 times better performance with the right TTs set, and still a six times better performance in the worst case.

Fig. 9(b) shows resource utilization. We see that the combination of high TT_{low} and high TT_{high} (Scenario 8) gives the best utilization (84.0%), whereas low values for TT_{low} and TT_{high} lead to the worst utilization (62.0% in Scenario 1). Still, compared to CBR which scored a maximum of 80.4% and a minimum of 51.8%, the rule-based approach generally achieves better results.

The percentage of all executed actions as compared to all possible actions that could have been executed is shown in Fig. 9(c). One observes that the greater the span between TT_{low} and TT_{high} is, the fewer actions have to be executed. Most actions (60.8%) are executed for Scenario 7 (span of only 5% between TT values), whereas least actions (5.5%) are executed for Scenario 3 (span of 60% between TT values). CBR almost always recommended exactly one (out of two possible) actions and hardly ever (in about 1% of the cases) recommended no action.

As violations are very low in general, the resource allocation efficiency is very similar to the utilization. The best value can be achieved with Scenario 8 (84.0%), the worst with Scenario 1 (62.0%). CBR achieves a RAE of at most 69.7% ($\alpha = 0.5$ at iteration 2), and at least 45.5% ($\alpha = 0.1$ at iteration 20).

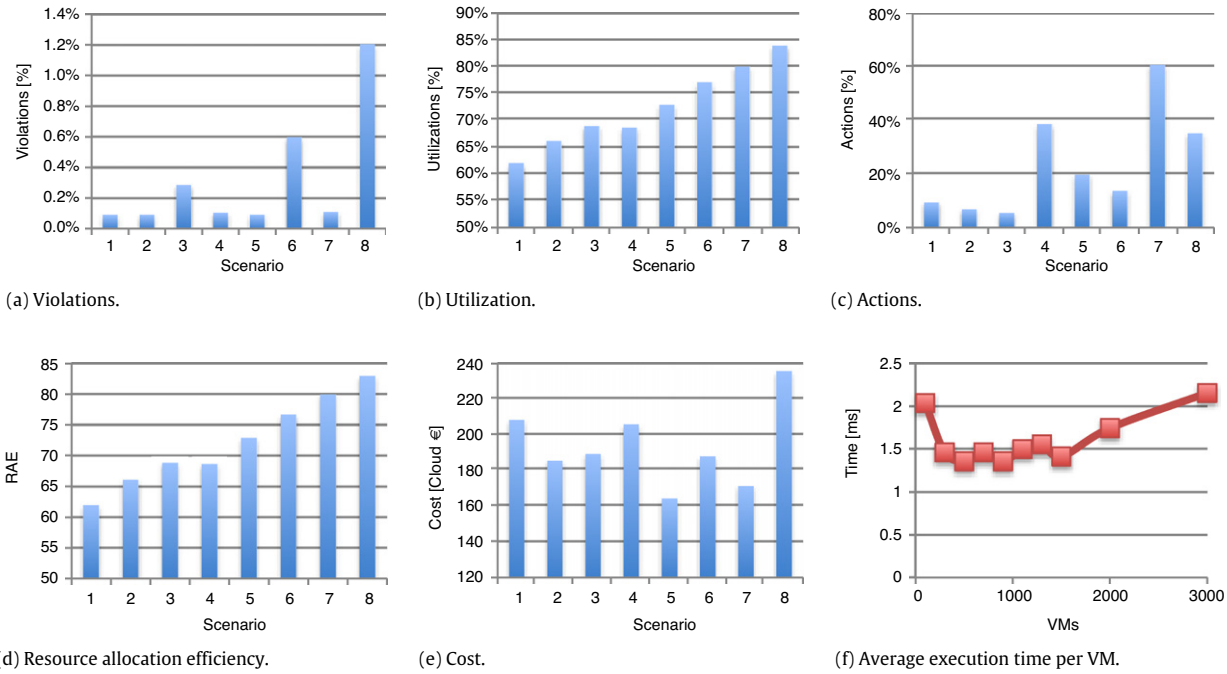
Fig. 8(e) shows the costs for each scenario using Eq. (8). The best trade-off between the three terms is achieved by Scenario 5 that has medium values for TT^r_{low} and TT^r_{high}. It has a very low violation rate of 0.0916%, a quite elaborate utilization of 72.9%, but achieves this with only 19.8% of actions. Scenario 7 achieves a better violation and utilization rate but at the cost of an action rate of 60.8%, and consequently has higher costs. The lowest cost value for CBR is 923.0 Cloud EUR, the highest 2985.3 Cloud EUR.

If the utility of the decision decreases for a certain time frame (as cost increases), the KB could determine the cost summand in Eq. (8) that contributes most to this decrease. For any resource r, if the term is $\mathfrak{p}$, then decrease TT^r_{high}. If the term is $\mathfrak{w}$, then increase TT^r_{low}. Otherwise, if the term is $\mathfrak{c}$, then widen the span of TT^r_{high} and TT^r_{low}, i.e., increase TT^r_{high} and decrease TT^r_{low}. We plan to investigate this in our future research.

As far as time performance and scalability are concerned, the performance tests are very encouraging. We executed 100 iterations from 100 to 3000 VMs. We performed every test twice and calculated the average execution time as well as the average time it took for the simulation engine to handle one VM. As shown in Fig. 9(f) the execution time per VM stays quite constant for up to 1500 VMs, and thus average execution time is about linear. For 3000 VMs, it took 647 s/100 = 6.47 s for one iteration to treat all VMs. The high time consumption per VM for 100 VMs in Fig. 9(f) is due to the initialization of the rule knowledge base which takes over-proportionally long for just a small number of VMs and does not weigh so much for more VMs.

CBR took 240 s for 50 VMs and 20 iterations. Thus, CBR took 240 s/20 = 12 s for one iteration to treat all VMs, which is twice as long as the rule-based approach takes, which even has 60 times more VMs. However, CBR implements learning features, which the rule-based approach currently does not, and could be sped up by choosing only specific cases to be stored in the KB.

Summarizing, the rule-based approach highly outperforms CBR with respect to violations (up to 82 times better results), actions, cost, and time performance. The rule-based approach also achieves better "best case" and better "worst case" results for the remaining performance indicators utilization and resource allocations efficiency. In more detail, seven out of eight scenarios were better than the worst CBR value for utilization, whereas only one scenario was better than the best CBR utilization value. Again, accumulating these results into cost, all rule-based scenarios outperform CBR by a factor of at least 4 (worst rule-based scenario (236) compared to the best CBR result (923)), which to a large extent is due to the huge number of violations that the rule-based



(a) Violations. (b) Utilization. (c) Actions.

(d) Resource allocation efficiency. (e) Cost. (f) Average execution time per VM.

Fig. 9. Violations, utilization, actions and utility for Scenarios 1–8, execution time for rule-based approach.

approach is able to prevent and the high number of actions it can save.

Consequently, we consider the rule-based approach as the better technique to deal with VM reconfiguration in Cloud Computing infrastructures, and we will focus the remaining part of this article on a deeper investigation and understanding of the rule-based approach by evaluating it with real world workload. A deeper investigation of synthetic workload also suggests the self-adaptation of the TTs from the rule-based approach. A successful self-adaptation has been presented in [42].

6.4. Applying and evaluating a bioinformatics workflow to the rule-based approach

As detailed in [43,44], bioinformatics workflows have gained a great need for large-scale data analysis. Due to the fact that these scientific workflows are very resource intensive and can take hours if not days to complete, provisioning them in an environment with fixed resources leads to poor performance. On the one hand, the workflow might run out of resources and thus may have to be restarted on a larger system. On the other hand, too many resources might be provisioned in order not to take risks of a premature abort, which may cause a lot of resources to be wasted. Thus, Cloud computing infrastructures offer a promising way to host these sorts of applications [10]. The monitoring data presented in this Section was gathered with the help of the Cloud monitoring framework Lom2His [3]. Using Lom2His we measured utilized resources of TopHat [45], a typical bioinformatics workflow application analyzing RNA-Seq data [46], for a duration of about three hours [9].

In the following we briefly describe the bioinformatics workflow in more detail. We here consider Next Generation Sequencing (NGS), a recently introduced high-throughput technology for the identification of nucleotide molecules like RNA or DNA in biomedical samples. The output of the sequencing process is a list of billions of character sequences called 'reads', each typically holds up to 35–200 letters that represent the individual DNA bases determined. Lately, this technology has also been used to identify and

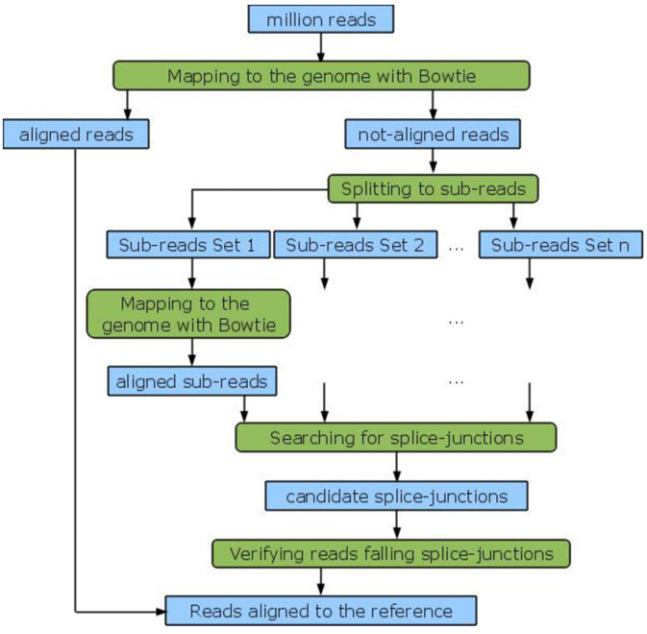
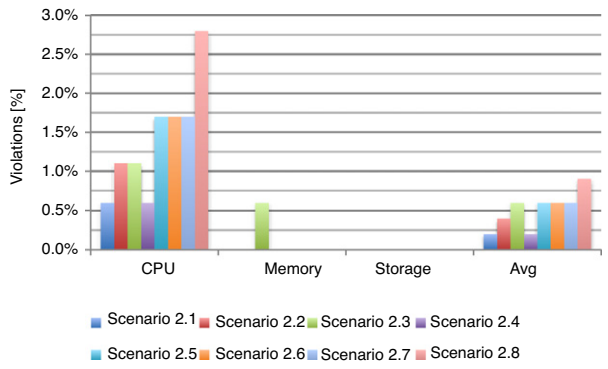


Fig. 10. Overview of the TopHat aligning approach.

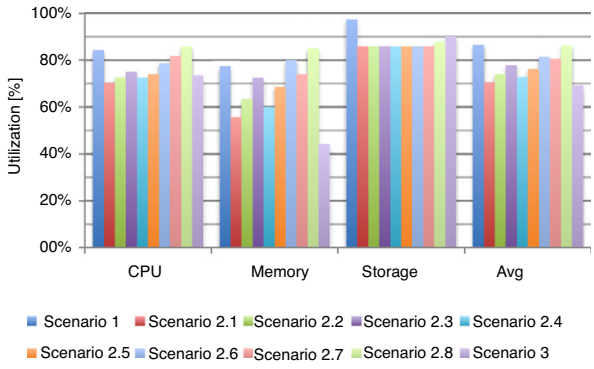
count the abundances of RNA molecules that reflect new gene activity. We use the approach, called RNA-Seq, as a typical example of a scientific workflow application in the field of bioinformatics.

At first, in the analysis of RNA-Seq data, the obtained sequences are aligned to the reference genome. The aligner presented here, TopHat [45], consists of many sub-tasks, some of them have to be executed sequentially, whereas others can run in parallel (Fig. 10). These sub-tasks can have different resource-demand characteristics: needing extensive computational power, demanding high I/O access, or requiring extensive memory size.

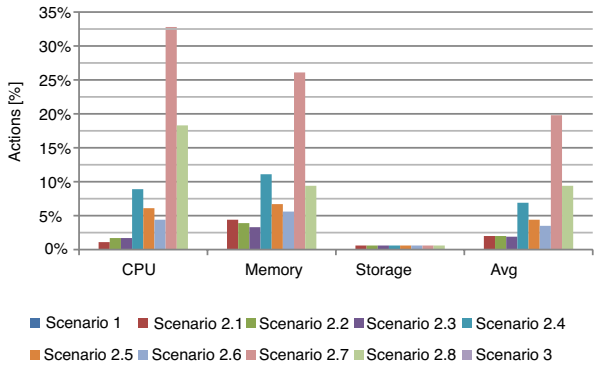
In Fig. 10, the green boxes represent simplified sub-tasks of the workflow application, whereas the blue boxes represent the data transferred between the sub-tasks. The first sub-task aligns input reads to the given genome using the Bowtie program [47].



(a) Violations.



(b) Utilization.



(c) Reconfiguration actions.

Fig. 11. Violations, utilization and reconfiguration actions for ten autonomic management scenarios using bioinformatics workflow.

Table 5
TopHat SLA.

Service Level Objective (SLO) name	SLO value
CPU power	≥20 000 MIPS
Memory	≥8192 MB
Storage	≥19 456 MB

Unaligned reads are then divided into shorter sub-sequences which are further aligned to the reference genome in the next sub-task. If sub-sequences coming from the same read were aligned successfully to the genome, that may indicate that this read was straddling a 'gap' in the gene, falling on a so-called splice-junction. After verification of candidate reads falling on splice junctions, these and the reads that were aligned in the first sub-task are combined to create an output with a comprehensive list of localized alignments.

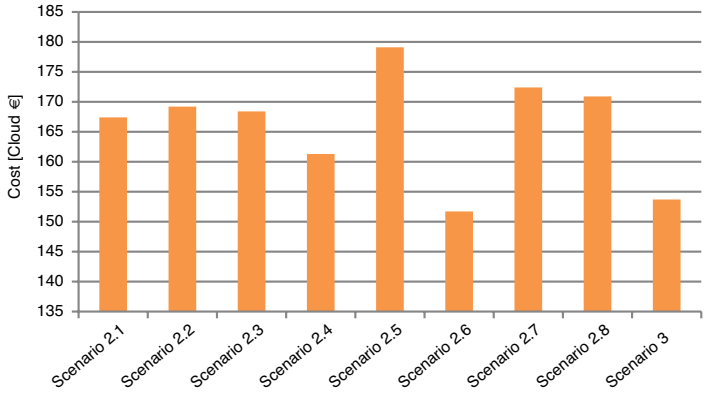
We demonstrate by simulation that the rule-based approach can guarantee the resource requirements in terms of CPU, memory and storage for the execution of the workflow in a resource-efficient way.

Therefore, we define the SLA shown in Table 5 for TopHat with the maximum amount of available resources on the physical machine on which we are executing it. The physical machine has a Linux/Ubuntu OS with a Intel Xeon(R) 3 GHz CPU, two cores, 9 GB of memory, and 19 GB of storage. For CPU power, we convert CPU utilization into MIPS based on the assumption that an Intel Xeon(R) 3 GHz processor delivers 10 000 MIPS for 100% resource utilization of one core, and linearly degrades with CPU utilization.

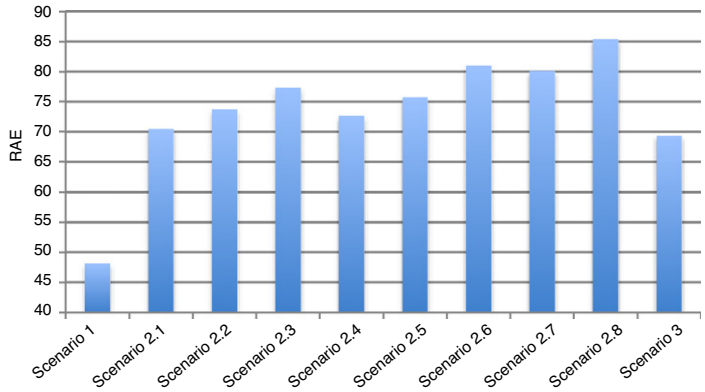
In order to validate our approach, we make three simulation categories, where we set up and manage our VMs differently: In the first category (Scenario 1) we assume a static configuration with a fixed initial resource configuration of the VMs. Normally, when setting up such a testbed as described in [9], an initial guess of possible resource consumption is done based on early monitoring data. From this data on, we assume quite generous resource limits. The first ten measurements of CPU, memory, and storage lie in the range of [140, 12 500] MIPS, [172, 1154] MB, [15.6, 15.7] GB, respectively. So we initially configured our VM with 15 000 MIPS, 4096 MB, and 17.1 GB, respectively. The second category subsumes several scenarios, where we apply our autonomic management approach to the initial configuration in the first category. The eight scenarios in this category depend on the chosen TTs. According to Table 4 we define these scenarios as Scenario 2.1, 2.2, . . . , 2.8, respectively. As the third category (Scenario 3), we consider a best case scenario, where we assume we have an oracle that predicts the maximal resource consumption that we statically set our VM configuration to. Moreover, according to the first measurements we decide to enforce a minimum of 1 MIPS CPU, 768 MB memory, and 1 GB storage.

As depicted in Fig. 11(a)–(c) one sees violations, utilization, as well as the number of reconfiguration actions, respectively, for every parameter (together with an average value) in the different scenarios. Generally, the bars are naturally ordered beginning from Scenario 1, over Scenarios 2.1, . . . , 2.8, ending with Scenario 3. The number of violations in Scenario 1 reach 41.7% for CPU and memory, and 49.4% for storage, which leads to an average of 44.3%. (For better visibility, these results have been excluded from Fig. 11(a).) Thus, we experience violations in almost half of the cases. This is especially crucial for parameters *memory* and *storage*, where program execution could fail, if it runs out of memory or storage, whereas for a violation of the parameter *CPU*, we would "only" delay the successful termination of the workflow.

With Scenarios 2.* we can reduce the SLA violations to a minimum. We completely avoid violations for *storage* in all sub-scenarios, as well as for *memory* in all but one sub-scenarios. Also *CPU* violations can be reduced to 0.6% for sub-scenarios



(a) Cost.



(b) Resource allocation efficiency.

Fig. 12. Resource allocation efficiency and cost for ten autonomic management scenarios using bioinformatics workflow.

2.1 and 2.4, and still achieve a maximum SLA violation rate of 2.8% with Scenario 2.8. The average SLA violation rate can be lowered to 0.2% in the best case. Scenario 3, of course, shows no violations. However, it is unlikely to know the maximum resource consumption before workflow execution.

As to the utilization of the resources, it is clearly higher when a lot of violations occur, so Scenario 1 naturally achieves high utilization. This is the case, because when a parameter is violated, then the resource is already fully used up, but even more of the resource would be needed to fulfill the needs. On the opposite, Scenario 3 naturally achieves low utilization, as a lot of resources are over-provisioned. Scenarios 2.* achieve a good utilization that is on average in between the two extremes and ranges from 70.6% (Scenario 2.1) to 86.2% (Scenario 2.8). Furthermore, we observe some exceptions to this "rule" when considering individual parameters. So, e.g., for memory we achieve a utilization of 85.0% with Scenario 2.8 or 80.0% with Scenario 2.6, which is higher than the utilization in Scenario 1 (77.4%). The same is true for CPU utilization rates of 85.5% as compared to 84.3% for the Scenario 1 and 2.8, respectively. Only for storage the utilization of all but one of the Scenarios 2.*, which is at 85.9%, is smaller than for Scenario 3 (90.1%).

A huge advantage of Scenarios 2.* is that they do not run into any crucial SLA violation (except for Scenario 2.3), but achieve a higher utilization as compared to Scenario 3. As to the reallocation actions, of course, Scenario 1 and 3 do not execute any, but also for the autonomic management in Scenarios 2.*, the amount of executed reallocation actions for most scenarios stays below 10%. Only Scenario 2.7 executes actions in 19.8% of the cases on average of the time. Five out of eight scenarios stay below 5% on average.

When it comes to the overall costs of the scenarios (cf. Fig. 12(a)), all 2.* scenarios approach the result achieved by the best case scenario 3. Scenario 1 sums up costs of 4493.6, and has therefore been omitted in the figure. Furthermore, the lowest cost is achieved using Scenario 2.6, which is even lower than the cost for Scenario 3. This is possible, because Scenario 2.6 achieves a very good utilization and SLA violation rate with a very low number of reallocation actions. Also resource allocation efficiency for Scenarios 2.* as shown in Fig. 12(b) achieves unambiguously better results than for Scenario 1 (RAE of 48.2%). Furthermore, all scenarios of the second category achieve a better RAE than the RAE of Scenario 3 (69.3%).

Thus, we conclude that by using the suggested autonomic management technique, we can avoid most costly SLA violations, and thus ensure workflow execution, together with a focus on resource-efficient usage. All this can be achieved by a very low number of time- and energy-consuming VM reallocation actions for many of the autonomic management scenarios.

7. Conclusion

The goal of this research field is to enact SLAs in a resource-efficient way with little human-based interaction in order to guarantee the scalability and strengthen the dynamic behavior and adaptation of the system. Autonomically governing Cloud Computing infrastructures is the investigated method leading to this goal.

In this paper we have hierarchically structured all possible reallocation actions, and designed, implemented, and evaluated two knowledge management techniques, Case Based Reasoning and a rule-based approach to achieve the aforementioned goal for one reallocation level, i.e., VM reconfiguration. After a comparison, we determined the rule-based approach to outperform CBR with

respect to violations and utilization, but also to time performance. Furthermore, we applied the rule-based approach to a real-world use case evaluating a scientific workflow from the area of bioinformatics. We showed by simulation that the rule-based approach can effectively guarantee the execution of a workload with unpredictably large resource consumptions.

The next step will be to move from simulation to a real Cloud testbed. Furthermore, the presented methods still involve some user-interaction for parameter tuning. Thus, it will be of great interest to autonomically determine crucial parameters of the presented methods and to adapt them based on current performance.

Another related field is the autonomic generation of IaaS SLA out of SaaS or PaaS SLAs. Theoretically, SaaS or PaaS applications can be perfectly set up on top of IaaS platforms. The crucial point is to extract an SLA for the IaaS parameters like bandwidth, storage, CPU power and memory that fit to SaaS/PaaS parameters like response time. It is obvious that response time directly relates to the mentioned IaaS parameters and user interaction. It is not that obvious, however, how this translation should take place. E.g., does the SLO "response time < 2 s" translate into "memory > 512 MB" and "CPU power > 8000 MIPS" or rather "memory > 4096 MB" and "CPU power > 1000 MIPS"? Once the autonomic governance of IaaS infrastructures is up and running, the autonomic translation of these SLAs will probably leverage the usage and usability of IaaS even more.

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. We also want to thank PawełP. Łabaj and David P. Kreil (Boku University Vienna) for providing us with the bioinformatics workflow application and Vincent C. Emeakaroha (TU Vienna) for providing monitoring data on it.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616. http://dx.doi.org/10.1016/j.future.2008.12.001.

[2] (FOSII)—Foundations of self-governing ICT infrastructures, March, 2012. http://www.infosys.tuwien.ac.at/linksites/FOSII.

[3] V.C. Emeakaroha, I. Brandic, M. Maurer, S. Dustdar, 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.

[4] I. Brandic, Towards self-manageable cloud services, in: S.I. Ahamed, et al. (Eds.), COMPSAC (2), IEEE Computer Society, 2009, pp. 128–133.

[5] B. Jacob, R. Lanyon-Hogg, D.K. Nadgir, A.F. Yassin, A Practical Guide to the IBM Autonomic Computing Toolkit, IBM Redbooks, 2004.

[6] M.C. Huebscher, J.A. McCann, A survey of autonomic computing—degrees, models, and applications, ACM Computing Surveys 40 (3) (2008) 1–28. http://doi.acm.org/10.1145/1380584.1380585.

[7] M. Maurer, I. Brandic, R. Sakellariou, Simulating autonomic SLA enactment in clouds using case based reasoning, in: ServiceWave 2010, Ghent, Belgium, 2010.

[8] M. Maurer, I. Brandic, R. Sakellariou, Enacting SLAs in clouds using rules, in: Euro-Par 2011, Bordeaux, France, 2011.

[9] V.C. Emeakaroha, P. Labaj, M. Maurer, I. Brandic, D.P. Kreil, Optimizing bioinformatics workflows for data analysis using cloud management techniques, in: The 6th Workshop on Workflows in Support of Large-Scale Science, WORKS11, 2011.

[10] N. Merchant, J. Hartman, S. Lowry, A. Lenards, D. Lowenthal, E. Skidmore, Leveraging cloud infrastructure for life science research laboratories: a generalized view, in: International Workshop on Cloud Computing at OOPSLA09, Orlando, USA, 2009.

[11] B. Khargharia, S. Hariri, M.S. Yousif, Autonomic power and performance management for computing systems, Cluster Computing 11 (2) (2008) 167–181.

[12] V. Petrucci, O. Loques, D. Mossé, A dynamic optimization model for power and performance management of virtualized clusters, e-Energy'10, ACM, New York, NY, USA, 2010, pp. 225–233. http://doi.acm.org/10.1145/1791314.1791350.

[13] M. Bichler, T. Setzer, B. Speitkamp, Capacity planning for virtualized servers, Presented at Workshop on Information Technologies and Systems, WITS, Milwaukee, Wisconsin, USA, 2006.

[14] G. Khanna, K. Beaty, G. Kar, A. Kochut, Application performance management in virtualized server environments, in: Network Operations and Management Symposium, 2006. NOMS 2006, 10th IEEE/IFIP, pp. 373–381, 2006. http://dx.doi.org/10.1109/NOMS.2006.1687567.

[15] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Sandpiper: black-box and gray-box resource management for virtual machines, Computer Networks 53 (17) (2009) 2923–2938. http://dx.doi.org/10.1016/j.comnet.2009.04.014.

[16] J. Rao, X. Bu, C.-Z. Xu, L. Wang, G. Yin, Vconf: a reinforcement learning approach to virtual machines auto-configuration, ICAC'09, ACM, New York, NY, USA, 2009, pp. 137–146. URL: http://doi.acm.org/10.1145/1555228.1555263.

[17] Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, Y. Coady, Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis, in: Cloud Computing, CLOUD, 2010 IEEE 3rd International Conference on, pp. 91–98, 2010. http://dx.doi.org/10.1109/CLOUD.2010.66.

[18] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, D. Pendarakis, Efficient resource provisioning in compute clouds via VM multiplexing, in: Proceeding of the 7th International Conference on Autonomic Computing, ICAC'10, ACM, New York, NY, USA, 2010, pp. 11–20. URL: http://doi.acm.org/10.1145/1809049.1809052.

[19] M. Mazzucco, D. Dyachuk, R. Deters, Maximizing cloud providers revenues via energy aware allocation policies, in: CLOUD 2010, pp. 131–138, 2010. http://dx.doi.org/10.1109/CLOUD.2010.68.

[20] M. Hoyer, K. Schröder, W. Nebel, Statistical static capacity management in virtualized data centers supporting fine grained QoS specification, in: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking, e-Energy'10, ACM, New York, NY, USA, 2010, pp. 51–60. URL: http://doi.acm.org/10.1145/1791314.1791322.

[21] D. Borgetto, H. Casanova, G.D. Costa, J.-M. Pierson, Energy-aware service allocation, Future Generation Computer Systems 28 (5) (2012) 769–779. http://dx.doi.org/10.1016/j.future.2011.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11000690.

[22] M. Stillwell, D. Schanzenbach, F. Vivien, H. Casanova, Resource allocation algorithms for virtualized service hosting platforms, Journal of Parallel and Distributed Computing 70 (9) (2010) 962–974. http://dx.doi.org/10.1016/j.jpdc.2010.05.006. URL: http://www.sciencedirect.com/science/article/pii/S0743731510000997.

[23] A. Nathani, S. Chaudhary, G. Somani, Policy based resource allocation in IaaS cloud, Future Generation Computer Systems 28 (1) (2012) 94–103. http://dx.doi.org/10.1016/j.future.2011.05.016. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11000987.

[24] S. Islam, J. Keung, K. Lee, A. Liu, Empirical prediction models for adaptive resource provisioning in the cloud, Future Generation Computer Systems 28 (1) (2012) 155–162. http://dx.doi.org/10.1016/j.future.2011.05.027. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11001129.

[25] G. Kecskemeti, G. Terstyanszky, P. Kacsuk, Z. Neméth, An approach for virtual appliance distribution for service deployment, Future Generation Computer Systems 27 (2011) 280–289. http://dx.doi.org/10.1016/j.future.2010.09.009.

[26] A. Paschke, M. Bichler, Knowledge representation concepts for automated SLA management, Decision Support Systems 46 (1) (2008) 187–205.

[27] G. Koumoutsos, S. Denazis, K. Thramboulidis, SLA e-negotiations, enforcement and management in an autonomic environment, in: Modelling Autonomic Communications Environments, 2008, pp. 120–125.

[28] R.M. Bahati, M.A. Bauer, 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, http://dx.doi.org/10.1109/ICAS.2008.47.

[29] Amazon elastic compute cloud (Amazon EC2), 2010. http://aws.amazon.com/ec2/.

[30] Rackspace cloud, March, 2012. http://www.rackspace.com/cloud/.

[31] Rightscale, 2012. http://www.rightscale.com/.

[32] SLA@SOI, March, 2012. http://sla-at-soi.eu/.

[33] M. Maurer, V.C. Emeakaroha, I. Brandic, J. Altmann, Cost-benefit analysis of an SLA mapping approach for defining standardized cloud computing goods, Future Generation Computer Systems 28 (1) (2012) 39–47. http://dx.doi.org/10.1016/j.future.2011.05.023. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11001051.

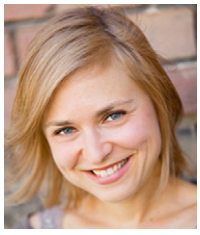
[34] I. Breskovic, M. Maurer, V.C. Emeakaroha, I. Brandic, S. Dustdar, Cost-efficient utilization of public sla templates in autonomic cloud markets, in: 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011, Melbourne, Australia, 2011.

[35] M.L. Massie, B.N. Chun, D.E. Culler, The ganglia distributed monitoring system: design, implementation and experience, Parallel Computing 30 (7) (2004) 817–840.

[36] L. Massoulie, J. Roberts, Bandwidth sharing: objectives and algorithms, in: INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings, vol. 3, IEEE, 1999, pp. 1395–1403. http://dx.doi.org/10.1109/INFCOM.1999.752159.

[37] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations: Proc. of a Symp. on the Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.
[38] B. Rochwerger, et al. The Reservoir model and architecture for open federated cloud computing, IBM Journal of Research and Development, 53 (4) (2009). URL: http://www.research.ibm.com/journal/rd/534/rochwerger.pdf.
[39] A. Aamodt, E. Plaza, Case-based reasoning: foundational issues, methodological variations, and system approaches, AI Communications 7 (1994) 39–59.
[40] M. Hefke, A framework for the successful introduction of KM using CBR and semantic web technologies, Journal of Universal Computer Science 10 (6) (2004).
[41] Drools, 2012. http://www.drools.org.
[42] M. Maurer, I. Brandic, R. Sakellariou, Self-adaptive and resource-efficient SLA enactment for cloud computing infrastructures, in: 5th International Conference on Cloud Computing, IEEE Cloud 2012, Honolulu, HI, USA, 2012.
[43] P. Romano, Automation of in-silico data analysis processes through workflow management systems, Briefings in Bioinformatics 9 (1) (2007) 57–68.
[44] D. Smedley, M.A. Swertz, K. Wolstencroft, G. Proctor, M. Zouberakis, J. Bard, J.M. Hancock, P. Schofield, Solutions for data integration in functional genomics: a critical assessment and case study, Briefings in Bioinformatics 9 (6) (2008) 532–544.
[45] C. Trapnell, L. Pachter, S.L. Salzberg, Tophat: discovering splice junctions with RNA-Seq, Bioinformatics 25 (9) (2009) 1105–1111. http://dx.doi.org/10.1093/bioinformatics/btp120.
[46] P.P. Łabaj, G.G. Leparc, B.E. Linggi, L.M. Markillie, H.S. Wiley, D.P. Kreil, Characterization and improvement of RNA-Seq precision in quantitative transcript expression profiling, Bioinformatics 27 (13) (2011) i383–i391. http://dx.doi.org/10.1093/bioinformatics/btr247.
[47] B. Langmead, C. Trapnell, M. Pop, S. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biology 10 (3) (2009) R25.

Michael Maurer is currently working as a project assistant at the Distributed System Group, Information Systems Institute, Vienna University of Technology (TU Wien). He studied the diploma program Applied Mathematics with a focus on Computer Science (first M.Sc.) and the master's program Computational Intelligence (second M.Sc.), which he finished both with distinction in November 2007 and April 2009 respectively. Furthermore, he studied at the City College of New York in New York USA during the winter term 2008. He has been working on the FoSII (Foundations of Self-governing ICT Infrastructures) project funded by the Vienna Science and Technology Fund (WWTF) since 2009. His research areas of interest include Cloud Computing, Autonomic Computing, Service Level Agreements, and Knowledge Databases. He is especially interested in the autonomic governing of Cloud Computing infrastructures.



Ivona Brandic is Assistant Professor at the Distributed Systems Group, Information Systems Institute, Vienna University of Technology (TU Wien). Prior to that, she was Assistant Professor at the Department of Scientific Computing, Vienna University. She received her Ph.D. degree from Vienna University of Technology in 2007. From 2003 to 2007 she participated in the special research project AURORA (Advanced Models, Applications and Software Systems for High Performance Computing) and the European Union's GEMSS (Grid-Enabled Medical Simulation Services) project. She is involved in the European Union's SCube project and she is leading the Austrian national FoSII (Foundations of Self-governing ICT Infrastructures) project funded by the Vienna Science and Technology Fund (WWTF). She is a Management Committee member of the European Commission's COST Action on Energy Efficient Large Scale Distributed Systems. From June to August 2008 she was visiting researcher at the University of Melbourne. Her interests comprise SLA and QoS management, service-oriented architectures, autonomic computing, workflow management, and large scale distributed systems (Cloud, Grid, Cluster, etc.).

Rizos Sakellariou is a Senior Lecturer in the School of Computer Science of the University of Manchester, where he teaches and leads a Research Laboratory with research interests in the wide area of High-Performance, Parallel and Distributed Computing. He has carried out research focusing primarily on issues related to parallelism and resource sharing, on a wide variety of topics (e.g., parallelizing compilers, performance modeling and prediction, multithreading, load balancing, workload and task scheduling, distributed query processing, grid information systems, service level agreements, adaptive and autonomic computing) and with a large number of collaborators. Currently, he is particularly interested in problems related to resource management and scheduling on any High-Performance and/or Parallel Computing environment (from multi-/many-core to grids and clouds), adaptive and autonomic computing, SLA-based and cloud computing.