Revealing the MAPE Loop for the Autonomic Management of Cloud Infrastructures

Michael Maurer, Ivan Breskovic, Vincent C. Emeakaroha, and Ivona Brandic
Institute of Information Systems, Vienna University of Technology
Argentinierstraße 8, 1040 Vienna, Austria
{maurer, breskovic, vincent, ivona}@infosys.tuwien.ac.at

ABSTRACT

Cloud computing is the result of the convergence of several concepts, ranging from virtualization, distributed application design, Grid computing, and enterprise IT management. Efficient management of Cloud computing infrastructures faces with the contradicting goals like unlimited scalability, provision of Service Level Agreements (SLAs), extensive use of virtualization, energy efficiency and minimization of the administration overhead by humans. Thus, autonomic computing seems to be one of the promising paradigms for the implementation of the management infrastructures for Clouds. However, currently available autonomic systems do not consider the characteristics of Clouds, e.g., virtualization layer, and thus are not easily applicable to Cloud infrastructures. In this paper we discuss first steps towards revealing the current MAPE (Monitoring, Analysis, Planning, Execution) loops for the application to Cloud infrastructures. We present novel techniques for the adequate monitoring of Clouds, discuss the approach for the knowledge management and present our solutions for facilitating SLA generation and management.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*; D.0 [**Software**]: General

General Terms

Autonomic Management, Cloud Middleware, MAPE Loop

Keywords

Cloud Computing, Autonomic Computing, SLA Management, Monitoring, Knowledge Management

1. INTRODUCTION

Cloud computing represents a novel paradigm for on-demand provision of ICT infrastructures, services, and applications. Thereby, resources are provisioned in predefined quality considering various functional and non-functional guarantees. Key concepts distinguishing Cloud Computing from other paradigms for the realization of large-scale distributed systems include (i) unlimited scalability of resources, (ii) sophisticated Service Level Agreement (SLA) management and generation, giving the customer guarantees on various non-functional aspects, and (iii) extensive use of virtualization technologies [1]. Many of the key concepts face with the contradicting goals, as, for example, unlimited scalability on the one hand and energy efficiency on the other. Scalability usually causes wastage of energy due to standby modes of devices and infrastructures. Thus, autonomic computing seems to be one of the promising solutions for the management of Cloud infrastructures by optimizing various (and maybe contradicting) goals as, for example, efficient resource usage, SLA management, virtualization and at the same time minimizing human interaction with the system and energy consumption.

Autonomic systems require high-level guidance from humans, but autonomically decide which steps need to be done to keep the system stable [2]. Such systems constantly adapt themselves to changing environmental conditions. Similar to biological systems, e.g., human body, autonomic systems maintain their state and adjust operations considering changing components, workload, external conditions, hardware, and software failures. Autonomic computing has been served as a promising concept for the infrastructure management in various areas, e.g., services, Grids, and SLA management [7, 4]. Autonomic control loop is known as MAPE [8], where (M) stands for monitoring of the managed elements, (A) for their analyzation, (P) for planning actions, and (E) for their execution. MAPE-K loop stores knowledge (K) required for decision-making in a knowledge base (KB).

However, existing autonomic frameworks, e.g., for Grids or SLA management, cannot be easily applied to Cloud computing infrastructures due to various reasons. For example, due to the virtualization layer, monitoring tools usually have to be configured on demand, distinguishing application based monitoring and resource based monitoring [3]. Energy efficiency requires novel techniques for the management of resources [5], while SLA generation requires advanced concepts for the management of the heterogeneous user base [6]. Thus, the traditional MAPE loop has to be revealed and tailored to Cloud specific solutions.

In Foundations of Self-Governing ICT Infrastructures (FoSII) project, we are developing novel techniques and methods for self-governing ICT infrastructures, and consequently applying the developed infrastructures for the self-managed Clouds [9]. One of the core research issues of the FoSII is the development of the appropriate autonomic loop suitable for the self-management of Clouds. Thus, in this paper, we propose an extended MAPE-K loop, called A-MAPE-K, including an *Adaptation* phase added to the traditional MAPE-K phases. Adaptation phase is necessary as a balance to the virtualization layer. During the *Adaptation (A)* phase, Cloud infrastructures, as well as applications to be deployed on the Clouds, are tailored and adapted. Moreover, we present novel concepts for the implementation of the *Monitoring* and *Knowledge Management* phases considering virtualization overhead.

Using the example of knowledge management, we present a sample prototype implementation for one part of the A-MAKPE-K loop. In order to efficiently manage running applications, we developed novel concepts for knowledge management where we learn from past experiences using the Case based reasoning (CBR) approach and where we successfully prevent future SLA violations and decrease resource wastage, i.e., increase resource utilization. We explain highlights of its implementation and evaluation.

The contributions of this paper are: (i) definition of the revealed MAPE-K loop suitable for the self-management of Clouds, including an additional *Adaptation* phase; (ii) application of the A-MAPE-K phases to the FoSII infrastructure; (iii) concepts for the implementation of the first three phases of the loop; and (iv) presentation and discussion of the first prototype for knowledge management in Clouds.

2. RELATED WORK

Since there is very little work on the autonomic loop management in Cloud computing infrastructures, we focus on the particular phases, including *SLA management & generation*, *monitoring* and *knowledge management* in Clouds.

SLA Management. Most of the current research work implying usage of SLAs considers SLA matching mechanisms which are based on semantic technologies. Oldham *et al.* [14] achieve autonomic SLA matching by utilizing semantic web technologies based on WSDL-S and OWL to enhance WS-Agreement specification. Dobson *et al.* [17] suggest producing a unified QoS ontology applicable to the QoS-based web service selection, QoS monitoring and QoS adaptation, as well as to several other scenarios. Ardagna *et al.* [15] introduce an autonomic Grid architecture with mechanisms for dynamically reconfiguring service center infrastructures in order to fulfill varying QoS requirements. Koller *et al.* [16] discuss autonomous QoS management using a proxy-like approach for defining QoS parameters that a service has to maintain during its interaction with a specific customer.

Monitoring. Fu *et al.* [11] propose GridEye, a service-oriented monitoring system with flexible architecture that is further equipped with an algorithm for prediction of the overall resource performance characteristics. The authors discuss how resources are monitored with their approach in Grid environment but they consider neither SLA management nor low-level metric mapping. Gunter *et al.* [12] present NetLogger, a distributed monitoring system, which can monitor and collect information of networks. Applications invoke NetLogger's API to survey the overload before and after some request or operation. However, it monitors only network resources. Wood *et al.* [13] developed a system, called Sandpiper, which automates the process of monitoring and detecting hotspots and remapping/reconfiguring VMs whenever necessary. Their monitoring system reminds ours in terms of goal: avoid SLA violation. Similar to our approach, Sandpiper uses thresholds to check whether SLAs can be violated. However, it differs from our system by not considering the mapping of low level metrics, such as CPU and memory, to high-level SLA parameters, such as response time for SLA enactment.

Knowledge Management. 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, as [19] on the performance of memory systems, or only deal with one or two specific SLA parameters. Petrucci [20] or Bichler [21] investigate one general resource constraint and Khanna [22] only focuses on response time and throughput. Paschke [4] et al. look into a rule based approach in combination with the logical formalism ContractLog. It specifies rules to trigger after a violation has occurred, but it does not deal with avoidance of SLA violations. Others inspected the use of ontologies as KBs only at a conceptual level. [23] viewed the system in four layers (i.e., business, system, network and device) and broke down the SLA into relevant information for each layer, which had the responsibility of allocating required resources. Again, no details on how to achieve this have been given. Bahati et al. [24] also use policies, i.e., rules, to achieve autonomic management. As in the other papers, this work deals with only one SLA parameter and a quite limited set of actions, and with violations and not with the avoidance thereof. Our KM system allows to choose any arbitrary number of parameters that can be adjusted on a VM.

All presented approaches consider either only one SLA parameter (e.g., execution time) and the whole MAPE loop, or they consider multiple SLA parameters but not all MAPE phases. To the best of our knowledge, none of the presented systems consider autonomic loops tailored to the needs and specifics of Cloud computing infrastructures.

3. BACKGROUND

In this section we discuss a motivating scenario for the development of the A-MAPE-K loop. Furthermore, we present SLA lifecycle, which should be supported by the autonomic loop, and finally we discuss the conceptional design of the A-MAPE-K loop.

3.1 Motivating Scenario

Table 1 depicts an SLA used to exemplify A-MAPE-K phases. We assume a PaaS scenario where SLAs specify guaranteed resources suitable for the application execution. The row *SLA Parameter* defines typical Service Level Objectives (SLOs) including *incoming bandwidth (IB)*, *outgoing bandwidth (OB)*, *storage (St)*, *availability (Av)* and *clock speed (Cs)*. The row *Value* specifies a concrete value with the according relational operator. SLAs are generated between the Cloud provider and user before the deployment of the application. In the following section, we will discuss the lifecycle necessary for the establishment and management of SLAs between the user and the provider.

3.2 SLA Lifecycle

Table 1: Sample SLA parameter objectives.

SLA Parameter	Value
Incoming Bandwidth (IB)	> 10 Mbit/s
Outgoing Bandwidth (OB)	> 12 Mbit/s
Storage (St)	> 1024 GB
Availability (Av)	$\geq 99\%$
Clock speed (Cs)	≥ 1000 MHz

We assume a typical Cloud use case where potential Cloud users deploy applications in a PaaS manner, as explained next. Service provider registers resources (i.e., platforms) to particular databases containing public SLA templates. Thereafter, Cloud users can lookup for Cloud services that they want to use for the deployment of applications. Similar to the provider, Cloud user has also an SLA template utilized for his 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, SLA contract can be negotiated and established and application can be deployed and executed.

Once the applications are deployed, the execution should be done in an autonomic way, minimizing users interaction with the system, optimizing energy consumption, and preventing violations of established SLAs. Resource management requires adequate monitoring techniques, wchih are used for application based SLA monitoring and deciding whether an SLA is violated or not. This is, however, far from trivial. Furthermore, in order to prevent SLA violations, knowledge management techniques are necessary. They are used to determine if applications can be migrated and virtual machines (VMs) and physical machines (PMs) (re)configured, migrated or switched off/on on demand in order to prevent SLA violations.

3.3 A-MAPE-K Loop Design

In this section we present how the aforementioned SLA lifecycle can be realized using the autonomic loop. We distinguish between system set up time where applications and infrastructure are tailored and *adapted*. Once the applications are deployed, we consider *monitoring*, *knowledge management* and *execution* phases during the application runtime. In this paper, in particular, we focus on the adaptation, monitoring, and knowledge management phases, as shown in Figure 1.

Adaptation As shown in Figure 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. 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 (see sample SLA parameter objectives, Table 1), or **missing attributes**. Therefore, a mechanism is required for the automatic adaptation

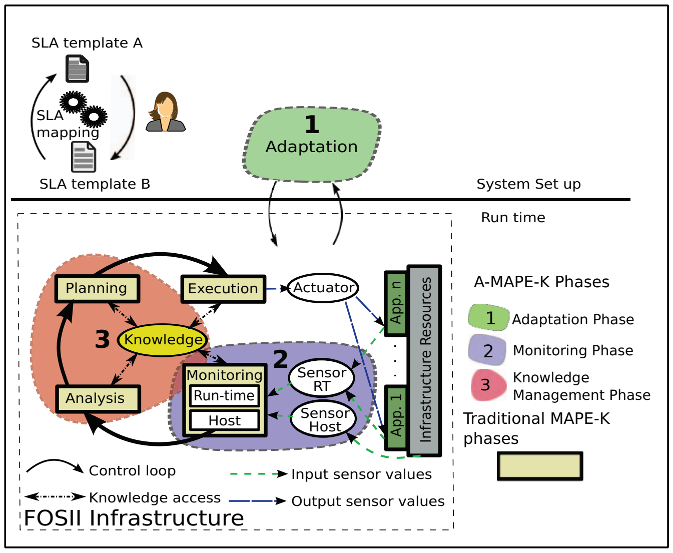

Figure 1: FoSII Infrastructure Overview

between different templates. Adaptation 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, *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.

Monitoring Clouds face with the problem of SLA parameters required by an application usually differing from the parameters measured by the monitoring tools. A typical application based SLA parameter is *system availability*, as depicted in Table 1. Current monitoring systems (e.g., ganglia [10]) facilitate monitoring only of low-level systems resources, such as *system up time* and *down time*. Thus, availability has to be calculated based on those low-level metrics. To achieve that, monitoring phase should comprise two core components, namely *host monitor* and *run-time monitor* (see Figure 1, part 2). The former is responsible for monitoring low-level resource metrics, e.g., system up time and down time directly delivered by the measurement tools (e.g., ganglia), whereas the latter is responsible for metric mapping, e.g., mapping of *system up time* and *down time* to *system availability* and consequently for the monitoring of SLA agreements.

Knowledge Management The term knowledge management means intelligent usage of measured data from the monitoring phase for the decision making process to satisfy SLA agreements while optimizing resources usage and consequently energy efficiency and minimizing user interactions with the system. In our approach, this includes not only 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. Since KM system uses monitoring information

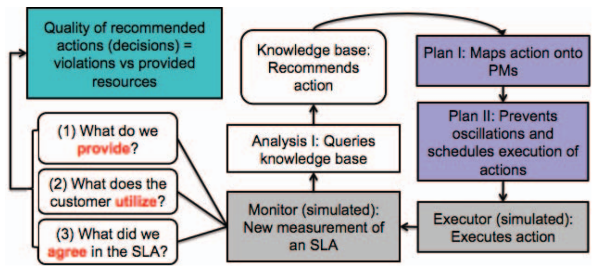
Quality of recommended actions (decisions) = violations vs provided resources
(1) What do we provide?
(2) What does the customer utilize?
(3) What did we agree in the SLA?
Knowledge base: Recommends action
Analysis I: Queries knowledge base
Monitor (simulated): New measurement of an SLA
Plan I: Maps action onto PMs
Plan II: Prevents oscillations and schedules execution of actions
Executor (simulated): Executes action

Figure 2: Generic simulation engine for knowledge management mapped to the A-MAPE-K loop

and directly recommends actions to prevent SLA violations and improve energy efficiency, we combine analysis and planning phases with the knowledge to the new *Knowledge Management Phase* (see part 3, Figure 1).

4. FOSII FRAMEWORK: IMPLEMENTATION EXAMPLE OF THE A-MAPE-K LOOP

In this section, we present the prototype implementation for the A-MAPE-K loop in the context of the FoSII infrastructure using the example of knowledge management based on the requirements discussed in Section 3.3.

4.1 Knowledge Management: CBR-Framework

Once the SLA parameters are measured as discussed in previous section, the knowledge management (KM) component should suggest reactive actions if necessary in order to prevent possible SLA violations.

We developed a KM technique-agnostic simulation engine able to evaluate the quality of various KM techniques by considering (i) efficient resource utilization, (ii) non-violation of SLAs, and (iii) efficient use of reallocation actions [5]. All three criteria are currently examined at the level of resource allocation to VMs, i.e., changing of VMs parameters on demand. Other stages would be the (re)allocation of VMs to PMs, which is part of our ongoing and future work.

The generic simulation engine traverses parts of the A-MAPE-K-cycle as depicted in Figure 2. *Monitor* and *Executor* parts are simulated (grey components in Figure 2). *Analysis I* and the *KB* are implemented using any arbitrary KM technique as long as it receive measurements and recommend actions (white components in Figure 2). Possible KM techniques are Case Based Reasoning (CBR), rule-based systems or default logic [25]. *Plan phases I* and *II* are executed using the FoSII framework. Possible reactive actions have to be pre-defined and are for the *Analysis phase* tied to parameters that can directly be tuned on a VM. Considering the SLA example from Table 1 with *Storage (St)*, possible actions in order to prevent SLA violations could be Increase/Decrease storage by 10%, 20%, etc., or to do nothing.

With the simulation engine we have so far thoroughly evaluated two KM techniques: CBR and rule-based approach. In the following we explain CBR approach.

CBR is the process of solving problems based on past experience [18]. 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. For Cloud Computing infrastructures a case consists of measured, i.e., actually used, and provided values of each resource. Together with the stored Service Level Objective (SLO) of every SLA parameter it can be retrieved whether an SLA violation occurred or not, cf. Figure 2. Examples for provided, utilized and agreed values are depicted in Table 2. As shown in Figure 2, we evaluate the quality of actions based on the provided, utilized and agreed values. At first, we deal with the provided value (1), which represents the amount of a specific resource that is currently used by the customer. Second, there is the amount of utilized/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.

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

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

Formalization. In the following we show some implementation details for the CBR approach as described in more detail in [5]. First we assume that each SLA has a unique identifier *id* and a collection of SLOs. SLOs are predicates of the form

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

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

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

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 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.

Similarity measurement. 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 [26] for the summation part this leads to the following equation

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

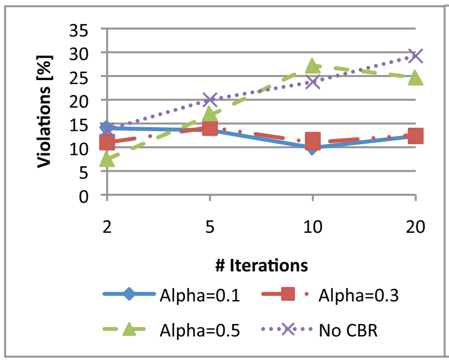

Figure 3: SLA violations

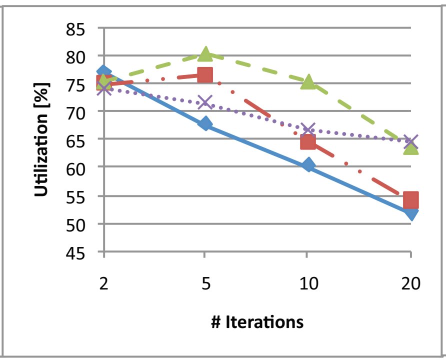

Figure 4: Utilization

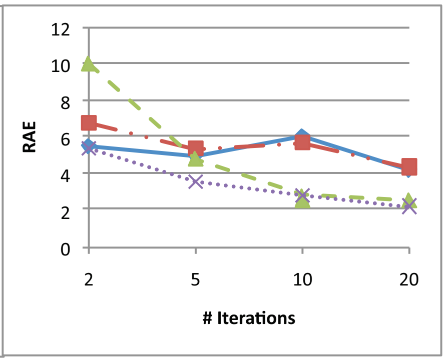

Figure 5: RAE

where $w = (w_{id}, w_{x_1}, \dots, w_{x_n})$ 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.

Utility function. Every action is evaluated by the first two criteria stated in the beginning of this section. Thus, 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 $0 \leq \alpha \leq 1$:

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

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

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

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

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

A utilization utility of 1 is retreived 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.

To compare the ratio of utilization, u and the number of SLA violations, v we define *resource allocation efficiency* (RAE) by

$$\text{RAE} = \frac{u}{v+1}, \qquad (7)$$

which reflects the above stated evaluation goals. High utilization yields high RAE, whereas a high number of SLA violations yields a low RAE, even if utilization is in normal range.

Evaluation. CBR has been evaluated over 2, 5, 10 and 20 iterations in terms of SLA violations, utilization, resource allocation efficiency and action ratio. Meaningful α values 0.1, 0.3, and 0.5 and evenly distributed weights w were used. As seen in Figures 3-5 the CBR results have been compared to "No CBR", which means that the initial values for every VM are statically set and not adapted during the iterations. In Figure 3 we see that for low $\alpha = 0.1, 0.3$ (yielding lower impact of utilization), violations can be reduced to up to a third as compared to no CBR. Figure 4 confirms that $\alpha = 0.5$ leads to higher utilization than no CBR. Figure 5 shows that RAE can be tripled with $\alpha = 0.1, 0.3$ as compared to the no CBR case. As to the action ratio (the percentage of all executed actions as compared to all possible actions that could have been executed), CBR recommended an action in 99% of all possible cases and only 1%it recommended to do nothing. Concerning time-performance, CBR is able to answer all recommendation requests for one iteration in 12s, thus taking 0.24s for one VM, which is a quite reasonable amount of time for decision making and learning. Concluding we see that CBR is able to improve the SLA violation, utilization and RAE rate, and achieves its learning peak after already 5 (for utilization) or 10 (for violations and RAE) iterations. Thus, one of the questions of future work is to see when to stop learning and which cases to retain, and whether another KM technique can even achieve better results, especially for the action ratio.

5. CONCLUSION

In this paper, we revealed the traditional MAPE loop for the autonomic management of Cloud infrastructures. We motivated the extension of the MAPE loop with an appropriate PaaS scenario where consumer and provider meet on demand, establish SLA agreements and where the Cloud management infrastructures autonomically manage the infrastructure considering various goals like energy efficiency or scalability. We extended the loop with the adaptation phase necessary as a balance to the virtualization layer. Moreover, we improved the monitoring phase considering low level systems metrics and high level SLA parameters. We combined the analysis and planing phase with the knowledge base and defined a new knowledge management (KM) phase. During the KM phase the observed monitoring information is fed into the KM system and reactive action is obtained which prevent SLA violations. For the KM phase we discussed a sample prototype implementation and concluded that Case Based Reasoning is a feasible KM technique to be used for the autonomic management of Cloud infrastructures.

Acknowledgment

The work described in this paper was partially supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII).

6. 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 Comp. Syst. 25(6): 599-616 (2009)

[2] J. O. Kephart and R. Das, *Achieving self-management via utility functions*, Internet Computing, January/February 2007 (vol. 11 no. 1) .

[3] V. C. Emeakaroha, R. N. Calheiros, M. A. S. Netto, I. Brandic, and C. A. F. De Rose. DeSVi: An Architecture for Detecting SLA Violations in Cloud Computing Infrastructures. CloudComp 2010, Barcelona, Spain October 25 - 28, 2010.

[4] A. Paschke and M. Bichler, *Knowledge representation concepts for automated SLA management*. Decision Support Systems, vol. 46, no. 1, pp. 187-205, 2008.

[5] M. Maurer, I. Brandic, R. Sakellariou: Simulating Autonomic SLA Enactment in Clouds Using Case Based Reasoning. ServiceWave 2010: 25-36

[6] M. Maurer, V. C. Emeakaroha, I. Brandic, J. Altmann. *Cost and Benefit of the SLA Mapping Approach for Defining Standardized Goods in Cloud Computing Markets*. UCC 2010, in conjunction with ICoAC 2010, December 14-16, 2010, Chennai, India.

[7] M. C. Huebscher and J. A. McCann, *A survey of autonomic computing - degrees, models, and applications*. ACM Comput. Surv., vol. 40, no. 3, pp. 1-28, 2008.

[8] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin, *A practical guide to the IBM Autonomic Computing toolkit*. IBM Redbooks, 2004.

[9] FOSII - Foundations of Self-governing ICT Infrastructures, http://www.infosys.tuwien.ac.at/linksites/fosii.

[10] M. L. Massie, B. N. Chun, D. E. Culler, The Ganglia Distributed Monitoring System: Design, implementation and experience, Parallel Computing Vol. 30 no. 7 pp. 817-840, 2004.

[11] W. Fu, Q. Huang, GridEye: A Service-Oriented Grid Monitoring System with Improved Forecasting Algorithm, Proceedings of the 5th International Conference on Grid and Cooperative Computing Workshops (GCCW'06), 2006.

[12] D. Gunter, B. Tierney, B. Crowley, M. Holding, J. Lee, Netlogger: A toolkit for distributed system performance analysis, in: Proceedings of the MASCOTS'00.

[13] T. Wood, P. J. Shenoy, A. Venkataramani, M. S. Yousif, Sandpiper: Black-box and gray-box resource management for virtual machines, Computer Networks 53 (17) (2009) 2923-2938.

[14] N. Oldham, K. Verma, Semantic WS-Agreement Partner Selection, In 15th Int. WWW Conf, pp. 697-706, ACM Press, 2006

[15] D. Ardagn, G. Giunta, N. Ingraffiam, R. Mir, B. Pernici, Qos-driven web services selection in autonomic grid environments, In OTM Conferences 2006, pp. 1273-1289, 2006

[16] B. Koller, L. Schubert, Towards autonomous SLA management using a proxy-like approach, Multiagent Grid Systems, Vol. 3 pp. 313-325, 2007

[17] G. Dobson, A. Sanchez-Macian, Towards Unified QoS/SLA Ontologies, Services Computing Workshops, 2006. SCW '06. IEEE, pp. 169-174, 2006

[18] Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches (1994).

[19] Khargharia, B., Hariri, S., Yousif, M.S.: Autonomic power and performance man- agement for computing systems. Cluster Computing 11(2), 167-181 (2008).

[20] Petrucci, V., Loques, O., Mosse, D.: A dynamic optimization model for power and performance management of virtualized clusters. In: e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. pp. 225-233. ACM, New York, NY, USA (2010)

[21] Bichler, M., Setzer, T., Speitkamp, B.: Capacity Planning for Virtualized Servers. Presented at Workshop on Information Technologies and Systems (WITS), Milwaukee, Wisconsin, USA, 2006 (2006)

[22] Application Performance Management in Virtualized Server Environments (2006), http://dx.doi.org/10.1109/NOMS.2006.1687567

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

[24] Bahati, R.M., Bauer, M.A.: Adapting to run-time changes in policies driving autonomic management. In: ICAS '08, Washington, DC, USA (2008).

[25] Maurer, M., Brandic, I., Emeakaroha, V.C., Dustdar, S.: Towards knowledge management in self-adaptable clouds. In: IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems. Miami, USA (2010).

[26] Hefke, M.: A framework for the successful introduction of KM using CBR and semantic web technologies. Journal of Universal Computer Science 10(6) (2004)