2012 IEEE Fifth International Conference on Cloud Computing

Self-Adaptive and Resource-Efficient SLA Enactment for Cloud Computing Infrastructures

Michael Maurer, Ivona Brandic
Distributed Systems Group
Vienna University of Technology
Vienna, Austria
{maurer, ivona}@infosys.tuwien.ac.at

Rizos Sakellariou
School of Computer Science
University of Manchester
Manchester, U.K.
rizos@cs.man.ac.uk

***Abstract*—Cloud providers aim at guaranteeing Service Level Agreements (SLAs) in a resource-efficient way. This, amongst others, means that resources of virtual (VMs) and physical machines (PMs) have to be autonomically allocated responding to external influences as workload or environmental changes. Thereby, workload volatility (WV) is one of the crucial factors that influence the quality of suggested allocations. In this paper we devise a novel approach for self-adaptive and resource-efficient decision-making considering the three conflicting goals of minimizing the number of SLA violations, maximizing resource utilization, and minimizing the number of necessary time- and energy-consuming reconfiguration actions. We propose self-adaptive rule-based knowledge management for autonomic VM reconfiguration considering the rapidness of changes in the workload, i.e., WV. We introduce a novel WV categorization and present cost and volatility based methods for self-tuning. We evaluate these methods by a large variety of synthetically generated workloads, and by real-world measurements gathered from an image rendering application and a scientific workflow for RNA sequencing. Evaluation shows that in most cases the self-adaptive approach outperforms the static approach.**

***Index Terms*—Cloud Computing, Autonomic Computing, Self-Adaptation, Service Level Agreement, Rule-based System, Knowledge Management, Resource Management.**

I. Introduction

The vision of Cloud computing is to provide computing power as a utility like electricity, gas, or water to a broad variety of customers [6]. To make Cloud computing a reliable means of computing, customers agree on so-called Service Level Agreements (SLAs) for certain non-functional QoS goals, the service price, and the penalty in case the provider violates the agreement. Cloud providers can offer their infrastructure as a service (IaaS), where the customer's application runs inside a virtual machine (VM). Governing such an infrastructure should happen autonomically to foster high scalability and to be able to react promptly and without human interaction to unforeseen external influences as workload changes.

Moreover, as energy costs of data centers are already very high [14], IaaS Cloud providers have a high incentive to minimize their energy consumption. As Cloud infrastructures are designed to host a large number of VMs, even slightly downsizing each of them – without causing SLA violations – might result into a big resource gain. This lower amount of provided resources can then be mitigated to reduce energy consumption, e.g., by powering off PMs that become unused.

Related work has observed the initial placement of VMs quite well, and some works also deal with the impact migrations have [26], [18]. However, VM sizes are normally assumed to be static (on Amazon, e.g., there exist only three types of VM sizes, which are not designed to change during runtime [1]) and changing their configuration has only been observed by few, as in [17]. The authors present a rule-based knowledge management (KM) approach that triggers actions to avoid under- and over-utilization of every resource based on threat thresholds (TTs). However, as with many approaches presented in related work, also this one depends on important parameters, i.e., TTs, that heavily influence performance, but neglects the configuration thereof. To achieve real autonomic governance of a Cloud infrastructure, it is crucial for any proposed approach to self-adapt its parameters to changing conditions in application usage, SLAs, and similar factors.

In this paper our prime focus is to investigate methods to autonomically set and adapt the TTs of the rule-based approach. We analyze several different methodologies. Whereas some methods set the TTs based on past performance, others rely on knowledge gathered from monitoring the workload itself. To achieve the latter, we introduce the notion of workload volatility (WV) and determine a way to calculate it and dynamically classify workload into WV classes. Furthermore, we investigate synthetically generating wide-spread workloads for Cloud applications. We use these and real-world workloads from an image rendering software, as well as a bioinformatics workflow for RNA sequencing [8], to evaluate our approach.

With this work we ultimately present a prototype for an autonomic SLA enactment and resource management tool for Cloud computing infrastructures on the level of VMs, whose advantages are manifold. Practically no a-priori learning is necessary and adaptation happens on the fly during execution. The approach automatically takes the different characteristics of various resource types into account. Finally, it is general in the sense that it does not only handle specific types of workload. It does not require specific domain knowledge nor is it specialized on only some domains like medical services or image rendering software. With its self-adaptability it is independent of any important parameters to be tuned.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 gives background informa-

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.55

tion about the autonomic loop and the rule-based approach. Thereafter, Section 4 details the autonomic parameter adaptation methods, and Section 5 evaluates them using various workloads. Finally, Section 6 concludes the paper.

II. Related Work

We have determined two different aspects to compare our work with: SLA and resource allocation management in Clouds, also related to KM; and self-adaptive algorithms in large-scale distributed systems.

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, as [13] on the performance of memory systems. Furthermore, Petrucci [20] and Bichler [4] investigate only one general resource constraint. A quite similar approach to our concept is provided by the Sandpiper framework [25], which offers black-box and gray-box resource management for VMs. Contrary to our approach, though, it plans reactions just after violations have occurred. The VCONF model [22] also pursues similar goals as presented in Section I, but depends on specific parameters, can only execute one action per iteration and neglects the energy consumption of executed actions. As to KM, Bahati et al. [3] also use 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. This is quite similar to the rule-based approach we base our work upon. However, their 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. Additionally, the regions are statically set and it is not investigated how to adapt them. Hoyer et al. [10] 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. [5] tackle the trade-off between consolidating VMs on PMs and turning off PMs on the one hand, and attaining SLAs for CPU and memory on the other. However, the authors assume a static setting and do not consider dynamically changing workloads. Summarizing, there has been a great deal of work on the different escalation levels, whereas VM configuration has not been observed yet neither its self-adaptation.

Secondly, Dutreilh et al. [7] investigate horizontal scaling, i.e., adding and removing VMs running an application server by using a load balancer, using a threshold-based and a reinforcement learning technique. However, the authors do not consider adapting the thresholds themselves via learning. Moreover, the authors determine problems with static thresholds as well as with determining good tuning for the reinforcement algorithms. The authors also state the importance of understanding the workload variation, but do not present a method how to deal with it. Kalyvianaki et al. [12] use Kalman filters for CPU resource provisioning for virtualized servers. They self-adapt their approach by using variances and covariances. Padala et al. [19] develop self-tuning controllers for multi-tier applications using control theory. Song et al. [23] use self-adaptation in the field of Cloud federations. Their algorithm selects tasks and allocates them by finding a trade-off between SLA adherence and resource utilization. This trade-off is represented by a parameter, which is optimized using a similar principle as the bisection method. For the optimization the benefit of a specific threshold is estimated by simulation. This estimation is executed several times until an adequate value is found. [21] apply genetic algorithms for decision making and self-reconfiguration, but on the network topology of remote data mirrors.

III. Background

In this section we describe the autonomic loop together with the rule-based knowledge management approach.

The presented management tool is an essential building block of the FoSII project [2], whose goal is to autonomically govern Cloud computing infrastructures. The management of the FoSII infrastructure relies on the autonomic control loop, which consists of the following phases: first, it monitors (M) the managed infrastructure with the help of sensors; second, it analyses the monitored data (A); third, it plans actions to execute (P); fourth, it executes them (E). The full loop is known as MAPE. The MAPE loop enhanced with a knowledge base (MAPE-K) [11] is the design paradigm for our approach. The monitoring information is gathered by the hardly intrusive and highly scalable Lom2His framework [9] and the execution of the actions is simulated by a KM-technique agnostic simulation engine [16].

In order to reduce the complexity of the NP-hard resource management problem in a Cloud, we hierarchically structure the problem into several so-called "escalation levels" [17]. This work is about the two lowest escalation levels, i.e., doing nothing and VM configuration. It is important to determine when to do nothing, since every reallocation action consumes time and energy. Thus, reallocation actions should only be recommended when necessary. Reallocation actions reset VM parameters like provided CPU power, storage, memory, or incoming/outgoing bandwidth. The rule-based approach achieves this and works as follows: The utilization of a resource is divided into three regions with the help of two threat thresholds (TTs), TT^r_{low} and TT^r_{high}. Region $+1$ ($ut^r < TT^r_{low}$) signifies a region, where the resource is over-provisioned. In region -1 ($ut^r > TT^r_{high}$) the resource is in danger of under-provisioning, or is already under-provisioned. In region 0 ($TT^r_{low} \leq ut^r \leq TT^r_{high}$) the resource is well provisioned. The central idea behind this design is that the ideal value called *target value* $tv(r)$ for the utilization of a certain resource is exactly in the center of region 0.

If the utilization value after some measurement leaves this region by using more (Region -1) or less resources (Region $+1$), we set the utilization back to the target value, i.e., we increase or decrease allocated resources so that the utilization

is again at

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

As long as the utilization value stays in region 0, no action will be recommended. A more detailed description of the rule-based approach can be found in [17].

All in all, we take a speculative approach: We try to allocate less than agreed as upper bound, but more than utilized without running into an SLA violation. Setting and adapting the mentioned TTs is the main focus of the remaining paper.

IV. Autonomic parameter adaptation

In this section we will explain how the autonomic adaptation and configuration of the autonomic manager works. Since the autonomic manager as presented in Section III is configured by threat thresholds, we will present their autonomic adaptation in this section. We will describe two basically different approaches: The first approach (cf. Section IV-A) is based on changes within a cost function, whereas the second approach (cf. Section IV-B) relies on changes in the workload.

A. Approaches based on the cost function

In this approach the autonomic adaptation of the TTs is based on the definition of the cost function in [17]. The general idea is that if cost has increased for some time, TTs should be adapted. Then two different subproblems have to be solved:

1) Determine the most appropriate TT(s) to adapt.
2) Determine for how much the chosen TT(s) should be adapted.

The used cost function is defined as

$$c(p, w, c) = \sum_r \mathfrak{p}^r(p^r) + \mathfrak{w}^r(w^r) + \mathfrak{a}^r(a^r), \qquad (1)$$

where, for a certain resource r, $\mathfrak{p}^r(p^r) : [0, 100] \to \mathbb{R}^+$ defines the costs due to the penalties that have to be paid according to the relative number of SLA violations (as compared to all possible SLA violations) p^r; $\mathfrak{w}^r(w^r) : [0, 100] \to \mathbb{R}^+$ defines the costs due to unutilized resources w^r; and $\mathfrak{a}^r(a^r) : [0, 100] \to \mathbb{R}^+$ the costs due to the executed number of actions a^r (as compared to the number of all possible actions).

During the *Analysis* phase the KB does not only observe the cost for one resource r, which naturally is defined as $c^r(p, w, c) = \mathfrak{p}^r(p^r) + \mathfrak{w}^r(w^r) + \mathfrak{a}^r(a^r)$, but also each individual component $\mathfrak{p}^r$, $\mathfrak{w}^r$, and $\mathfrak{a}^r$ for each resource. If the cost has increased for a resource over a certain period of time (called *look-back horizon* k and defined later in this section), the KB starts to investigate which of the components caused this increase.

Subproblem 1 (Selecting TTs). To solve subproblem 1, at first the most problematic cost factor has to be determined. From this, we can relate to a specific TT increase/decrease action. To achieve this one can basically imagine two different methodologies: Either, the maximum cost parameter of the current iteration, or the parameter with the maximum increase in the last k iterations is chosen.

Since our cost function c^r works by relative and not total costs, the first method would yield the following problem: Suppose that no violation has occurred for 10 iterations. Thus, $p^r = 0$ at iteration 10. At iteration 11, though, a violation occurs which makes $p^r = 1/11$. In the following iterations, where $p^r = 1/12, 1/13, 1/14, \ldots$ (if no further violations occurs) $\mathfrak{p}^r$ could be easily greater than $\mathfrak{w}^r$ and $\mathfrak{a}^r$ as violations are usually punished more severely than wastage or actions. Thus, for these iterations the algorithm would always decide to act based on violations, even though violations are not occurring any more in the same time.

Let $p^{r,t}$ signify the relative amount of violations at iteration t, and let $w^{r,t}$ $a^{r,t}$ be defined similarly. Then, since an increase in, e.g., violations $p^{r,t}$ occurs iff $\mathfrak{p}^{r,t}$ is strongly monotonically increasing, we choose to opt for the second methodology. According to a look-back horizon k we calculate the difference between the current cost and the minimum cost of the last k iterations. The maximum of these differences then points to the cost summand (arg) that needs attention:

$$\arg\max(\mathfrak{p}^{r,t} - \min_{1\le j\le k}(\mathfrak{p}^{r,t-j}), \mathfrak{w}^{r,t} - \min_{1\le j\le k}(\mathfrak{w}^{r,t-j}), \mathfrak{a}^{r,t} - \min_{1\le j\le k}(\mathfrak{a}^{r,t-j})). \qquad (2)$$

This results into three different cases, where either the $\mathfrak{p}$, $\mathfrak{w}$, or $\mathfrak{a}$ terms yield the maximum. (We omit cases where some arguments of the maximum function are equal. In such a case, the order to choose the arg max is $\mathfrak{p}$ over $\mathfrak{w}$ over $\mathfrak{a}$. We prioritize like this, because we assume that penalties incur higher costs than wastage, and wastage incurs higher costs than reconfiguration actions.) We define three options which TT(s) to increase or decrease.

- *Option A:*
  1) $\mathfrak{p}^{r,t} - \min_{1\le j\le k}(\mathfrak{p}^{r,t-j})$ is maximal: Decrease TT^r_{high} and TT^r_{low}.
  2) $\mathfrak{w}^{r,t} - \min_{1\le j\le k}(\mathfrak{w}^{r,t-j})$ is maximal: Increase TT^r_{low}.
  3) $\mathfrak{a}^{r,t} - \min_{1\le j\le k}(\mathfrak{a}^{r,t-j})$ is maximal: Decrease TT^r_{low} and increase TT^r_{high}.
- *Option B:*
  1) $\mathfrak{p}^{r,t} - \min_{1\le j\le k}(\mathfrak{p}^{r,t-j})$ is maximal: Decrease TT^r_{high} and TT^r_{low}.
  2) $\mathfrak{w}^{r,t} - \min_{1\le j\le k}(\mathfrak{w}^{r,t-j})$ is maximal: Increase TT^r_{high} and TT^r_{low}.
  3) $\mathfrak{a}^{r,t} - \min_{1\le j\le k}(\mathfrak{a}^{r,t-j})$ is maximal: Decrease TT^r_{low} and increase TT^r_{high}.
- *Option C:*
  1) $\mathfrak{p}^{r,t} - \min_{1\le j\le k}(\mathfrak{p}^{r,t-j})$ is maximal: Decrease TT^r_{high}.
  2) $\mathfrak{w}^{r,t} - \min_{1\le j\le k}(\mathfrak{w}^{r,t-j})$ is maximal: Increase TT^r_{low}.
  3) $\mathfrak{a}^{r,t} - \min_{1\le j\le k}(\mathfrak{a}^{r,t-j})$ is maximal: Decrease TT^r_{low} and increase TT^r_{high}.

The difference between options A and B is that if the $\mathfrak{w}$ term causes the maximum, it will increase both low and high TTs in option B, whereas it will only increase TT_{low} in option A. The main feature of option C is that it only decreases TT_{high} (instead of also decreasing TT_{low}). So option B and even more

option A could be seen as more cautious as far penalties for SLA violations are concerned than option C.

Moreover, we present a fourth methodology, *option D*, differing from the former three ones. This methodology does not only consider the maximum cost summand increase, but handles all cost parameters that show an increase, but only for the recent iteration. This may promise that the actual situation of which parameter needs to be adapted is assessed more precisely. Thus, one can distinguish seven different cases:

1) $\mathfrak{p}^r$ increased: Decrease TT^r_{high}.
2) $\mathfrak{w}^r$ increased: Increase TT^r_{low}.
3) $\mathfrak{a}^r$ increased: Decrease TT^r_{low}, increase TT^r_{high}.
4) $\mathfrak{p}^r$ and $\mathfrak{w}^r$ increased: Increase TT^r_{low}, decrease TT^r_{high}.
5) $\mathfrak{p}^r$ and $\mathfrak{a}^r$ increased: Decrease TT^r_{low}.
6) $\mathfrak{w}^r$ and $\mathfrak{a}^r$ increased: Increase TT^r_{high}.
7) $\mathfrak{p}^r$ and $\mathfrak{w}^r$ and $\mathfrak{a}^r$ increased: Choose the two factors with the highest increase and act according to the cases 4-6.

Subproblem 2 (Adapting TTs). After subproblem 1 has been solved, for subproblem 2 it is important to determine the value by how much the respective TT(s) should be moved. Again, one could imagine several techniques to determine a good value for the TTs as Case Based Reasoning (adapting the approach as described in [16]), or using fixed or random increasing/decreasing steps. Observing that for the TTs the following inequalities must hold

$$0\% < TT_{low} < TT_{high} < 100\%, \tag{3}$$

we choose to use the following approach. If we need to decrease TT_{low} or increase TT_{high}, we set it to a certain fraction $1/\alpha < 1$ of the distance from TT_{low} to 0, and from TT_{high} to 100, respectively, expressed as

$$TT^{r,t+1}_{low} = TT^{r,t}_{low} - \frac{TT^{r,t}_{low}}{\alpha} \tag{4}$$

$$TT^{r,t+1}_{high} = TT^{r,t}_{high} + \frac{100 - TT^{r,t}_{high}}{\alpha}. \tag{5}$$

(Superindex t indicates the time iteration for which the respective TT is valid. It is omitted, if not relevant.) If we need to increase TT_{low} or decrease TT_{high}, we shrink the distance d between TT_{low} and TT_{high} to $\frac{d(\alpha-1)}{\alpha}$ by moving the TT in question towards the other, i.e.,

$$TT^{r,t+1}_{low} = TT^{r,t}_{low} + \frac{TT^{r,t}_{high} - TT^{r,t}_{low}}{\alpha} \tag{6}$$

$$TT^{r,t+1}_{high} = TT^{r,t}_{high} - \frac{TT^{r,t}_{high} - TT^{r,t}_{low}}{\alpha}. \tag{7}$$

This especially makes sure that Eq. (3) also holds in this situation. When both TT_{low} and TT_{high} are to be increased and decreased, respectively, simultaneously (cf. case 4 in option D), we have to set $\alpha > 2$ in order not to violate Eq.(3).

Summarizing both subproblems, the graphs in Figure 1 show how the TTs behave for the different options A-C according to the following scenario: All options start with $TT_{low} = 50\%, TT_{high} = 75\%$. At iteration 2 we encounter a maximum in penalties, at iteration 4 a maximum in wastage and at iteration 6 a maximum in actions.

B. Approach based on workload volatility

As an alternative to the cost function dependent approach, we investigate an approach depending on the change in the workload, i.e., the workload volatility.

We define *workload volatility* ϕ as the intensity of change in the measured workload traces of a certain resource. We calculate this intensity as the percentage relating the current value of the workload $m^{r,t}$ to the previous one $m^{r,t-1}$, i.e.,

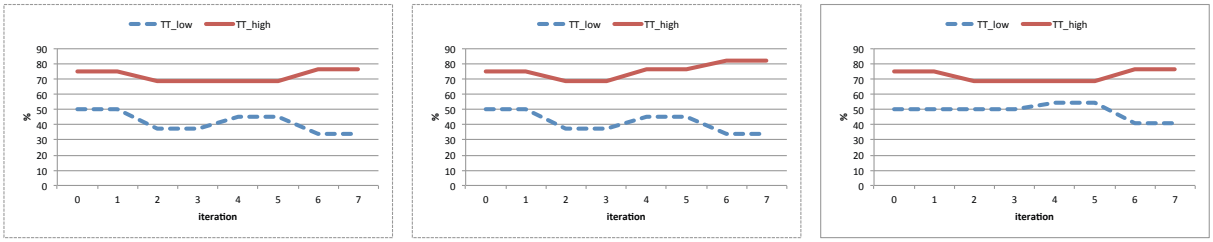
$$\phi^{r,t}(m^{r,t}, m^{r,t-1}) = |(\frac{\max(m^{r,t}, r_{min})}{\max(m^{r,t-1}, r_{min})} - 1) \cdot 100|$$

for $t \geq 1$ and $r_{min} > 0$. The variable r_{min} stands for the lower bound for a certain resource stated in the Service Level Objective (SLO). E.g., we have $r_{min} = 10$ for the SLO "$10GB \leq storage \leq 1000GB$". This amount will always be provided, even if an application uses less. So measurements below this value should not influence the behavior of the system, neither the classification into a WV class. To give an example for $r = storage$, let us assume that $m^{r,t} = 20, m^{r,t-1} = 15$. We would get $\phi^{r,t}(m^{r,t}, m^{r,t-1}) = 33.\dot{3}\%$. If at the next iteration we have $m^{r,t+1} = 18$, then $\phi^{r,t+1}(m^{r,t+1}, m^{r,t}) = 10\%$.

This is useful, because a problem inherent in options A-C is that the new parameter k to be tuned is introduced. Its relevance to WV is the following: When WV is low, a long look-back horizon is helpful, because a short one would trigger more TT adaptation situations, which in reality are just insignificant changes in workload. On the opposite, when WV is high, changes can get very fast very significant, and thus a short look-back horizon should be favored.

For this methodology, we introduce WV classes, into which we automatically categorize workload on the fly. We define the following WV classes: LOW, MEDIUM, MEDIUM_HIGH, and HIGH. Algorithm 1 dynamically decides to which WV class a specific workload trace belongs. Dynamically means that the classification might change at every iteration, if the workload behavior changes significantly. Significant in this context means that the current value for WV is compared to the recent behavior of the workload. Only if the maximum value for the WV from recent and current behavior falls into a different category, the classification is altered. From the second iteration on, the algorithm first calculates ϕ and determines the maximum value in ϕQ, which is a queue of size $\phi Q\_maxsize$ (lines 2-7). The method $addLast()$ adds the input element as last element to the queue, whereas the method $remove()$ removes the first element of the queue. Lines 9-18 classify the workload according to the found maximum element of the queue. An ϵ is added to this comparison in order to hinder small statistical outliers from altering the classification outcome. Table I summarizes all constants used for the evaluation.

Based on this classification the following two options E and F alter their behavior accordingly. *Option E* chooses a "good" set of TTs from a-priori evaluation for different WV classes. This can be tested offline, and altered if specified in the SLA. E.g., for high-risk applications both TTs could be lowered, whereas for energy-aware applications, the TTs could



(a) TT example for option A (b) TT example for option B (c) TT example for option C

Figure 1: TT examples for options A - C

Algorithm 1 On-the-fly Classifying of Workload into its Workload Volatility Class

Input: $r, m^{r,t}, m^{r,t-1}, \phi Q^r$
Output: Workload volatility class

if $t \geq 1$ **then**
  {Calculate ϕ and determine maximum in ϕQ^r}
  $\phi Q^r.addLast(\phi^{r,t}(m^{r,t}, m^{r,t-1}))$
  if $\phi Q^r.size() > \phi Q\_maxsize$ **then**
    $\phi Q^r.remove()$
  end if
  $\phi Q^r_{max} \leftarrow \max(\phi Q^r)$

  {Classify workload volatility}
  if $\phi Q^r_{max} \leq$ LOW_THRESHOLD $+ \epsilon$ **then**
    return LOW
  else if $\phi Q^r_{max} \leq$ MEDIUM_THRESHOLD $+ \epsilon$ **then**
    return MEDIUM
  else if $\phi Q^r_{max} \leq$ MEDIUM_HIGH_THRESHOLD $+ \epsilon$ **then**
    return MEDIUM_HIGH
  else if $\phi Q^r_{max} \leq$ HIGH_THRESHOLD $+ \epsilon$ **then**
    return HIGH
  end if
end if

Parameter	Value
LOW_THRESHOLD	10
MEDIUM_THRESHOLD	50
MEDIUM_HIGH_THRESHOLD	75
HIGH_THRESHOLD	100
$\phi Q\_maxsize$	10
ϵ	4

Table I: Parameters used for Algorithm 1

be increased for all workloads. For our case, Table II shows the TTs for the mentioned volatility classes.

Also from a-priori experience, *option F* chooses the best option with its best k according to the best result in the corresponding WV class. As will be seen in Section V, the best results for every WV class can be achieved by the options captured in the right-hand side of Table II.

V. Evaluation

In this section we evaluate the presented methods for autonomic TT configuration. We will first describe the used synthetic and real-world workloads, and then present their in-depth evaluation.

WV	*Option E)* TT_{low}	TT_{high}	*Option F)* Choose Option
LOW	70%	90%	C), $k = 5$
MEDIUM	45%	70%	A), $k = 20$
MEDIUM_HIGH	30%	60%	A), $k = 5$
HIGH	20%	50%	A), $k = 2$

Table II: A-priori defined TTs and options based on workload volatility classes for options E) and F)

A. Workloads

In this subsection we shortly describe the used workloads. We will first present the synthetic workload and then two real-world workloads. All of them show static behavior, as well as rapid, and also continuous changes.

The workload generator originally developed in [16] is intended to generate very general workloads for IaaS platforms. For one parameter, the workload generation is briefly sketched as follows: After the initial value is drawn from a Gaussian distribution an up- or down-trend is randomly drawn, as well as a duration of this trend, 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, consequently, not need a lot of dynamic adaptations. Large $iEnd$ values, on the contrary, need the enforcement mechanisms to be very elastically tuned. For the evaluation we defined and tested LOW, MEDIUM, MEDIUM_HIGH and HIGH WV scenarios with $iEnd = 10\%, 50\%, 75\%$, and 100%, respectively. As a minimum change we set $iBegin = 2\%$ for all scenarios.

Additionally, we tested real monitoring data gathered using the mentioned Cloud monitoring framework Lom2His. First, we measured some execution runs of the image-rendering application PovRay [9]. The workloads for PovRay contain 13 independent measurements from two categories. Due to the lack of space, we will present the most interesting three results from each of the categories. Measurements from the first category stem from rendering a fish jumping out of and

into water. We will tag these workloads POV_F1 to POV_F3. The workloads of the second category stem from rendering frames for a zoom-up on a box with other boxes inside, which we will call POV_B1 to POV_B3. The different runs within a category just differ in the image resolution.

Second, we evaluated our approach with measurements gained from the execution of a bioinformatics scientific workflow application. We measured utilized resources of TopHat [24], a typical bioinformatics workflow application analyzing RNA-Seq data [15], for a duration of about three hours [8].

B. TT adaptation using synthetic workloads

In this subsection we evaluate the six options A-F presented in Section IV using synthetic workload. As a quality measure, we will use the cost function defined by Eq.(1) with $\mathfrak{p}^r(p) = 100p, \mathfrak{w}^r(w) = 5w$, and $\mathfrak{a}^r(a) = a$ for all r, and for all adaptation options we set $\alpha = 4$ as used in Eqs. (4)-(7).

r_{min}		SLA parameter		r_{max}
1 GB	$\leq$	storage	$\leq$	1000 GB
1 Mbit/s	$\leq$	incoming bandwidth	$\leq$	20 Mbit/s
1 Mbit/s	$\leq$	outgoing bandwidth	$\leq$	50 Mbit/s
1 MIPS	$\leq$	CPU power	$\leq$	100 MIPS
8 MB	$\leq$	memory	$\leq$	512 MB

Table III: SLA for synthetic workloads

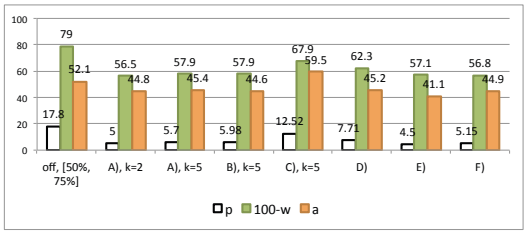
Every simulation run consists of 100 iterations. The SLA for the synthetic workloads is presented in Table III. Results of the simulation runs can be seen in Figures 2 - 4. In all Subfigures 2-5(a) we present p, $100 - w$, a for every simulation run. The specifics of each run are explained below each group of three bars: At first the adaptation option is stated, or "off", if none is used. Adaptation options also show k where applicable. All autonomic TT experiments have been conducted with $TT_{low} = 50\%$ and $TT_{high} = 75\%$ initially set (we will refer to this as the *standard case*), unless stated otherwise. This was chosen based on the evaluation in [17], as this setting brought best results for a LOW_MEDIUM WV class with $iEnd = 18\%$. For compact notation a TT pair is written as $[TT_{low}, TT_{high}]$. In all Subfigures 2-5(b) we show the cost $c(p, w, c)$ with the parameters as defined above.

The first three (group of) bars in Figure 2 represent static TT configurations evaluated in [17]. The goal of the autonomic TT management is to achieve costs that are as low or lower than the costs resulting from a static TT configuration. We see that the best static result in terms of costs can be achieved setting $TTs = [70\%, 90\%]$, and the cost for the standard case is 159. This value is beaten (or attained) by evaluated options A for $k \leq 25$, C for $k = 2, 5$ with the standard TT pair, C for all evaluated k with the best (a-priori unknown) TT pair, and options E and F. The best case is attained by options C with the best TT pair, and by option E.

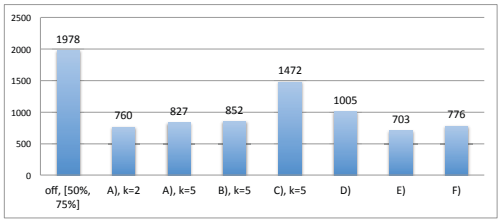
For the MEDIUM WV class we deduce from Figure 3 that options A for $k \geq 15$, E and F beat the static TT scenario. On the contrary, option C achieves the worst results by far.

Due to space limitations, we omit the graphs of the MEDIUM_HIGH WV class, which are quite similar to those of the HIGH WV class. Evaluation shows that all options except option C beat the results from the standard case. Option E achieves the best result.

As far as the HIGH WV class is concerned (cf. Figure 4), all options beat the results from the "standard case". From these, again option C still achieves the worst results, and again option E results into lowest costs.



(a) Violations p, utilization $100 - w$, actions a



(b) Cost $c(p, w, a)$

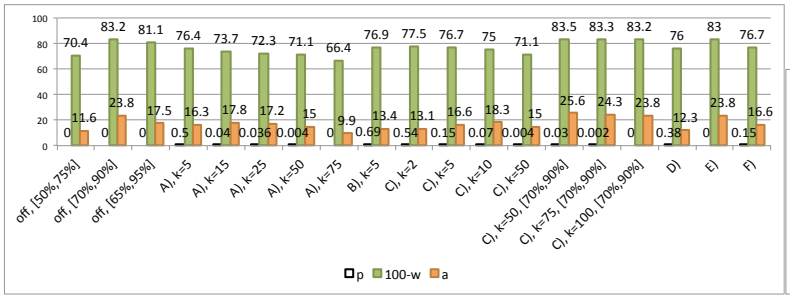
Figure 4: Evaluation results for HIGH WV class

Generally, autonomic adaptation works best for workloads with higher volatility and quite acceptable for workloads with lower volatility. We also see that option C for $k = 5$ generally achieves worst results except for low WV. This is explained by the fact as stated in Section IV that option C is less cautious than other options with respect to SLA violations. These violations, naturally, have a higher impact with higher WV. Option B for $k = 5$ achieves the worst result for LOW WV, and only outperforms the standard case for MEDIUM_HIGH and HIGH WV classes. Nevertheless, options E and F always outperform the standard case, and achieve best or very good results, and there is always a k for option A such that it also outperforms the standard case. The best cases for each WV class have been resembled in option F.

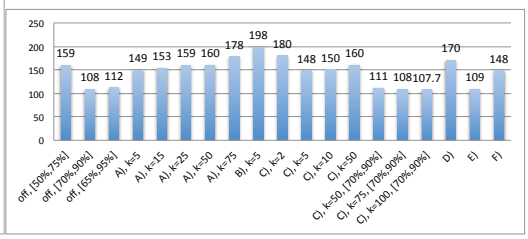
C. TT adaption using real-world workloads

This section presents the evaluation of two real-world workloads categories. One important point to observe with these workloads is that they do no longer fall into the same WV class for all the resources.

The SLA for the POVRay application is depicted in Table IV. As we have seen that in the previous subsection options E and F always outperform the standard case, we chose only these options for further evaluation. As can be seen in Table V (AM describes whether the autonomic manager is turned on or off), we remark that for POV_F* options E and F always outperform the standard case with partially big cost improvements up to 48% (for POV_F9), while the better option is not clearly the one or the other. For POV_B*

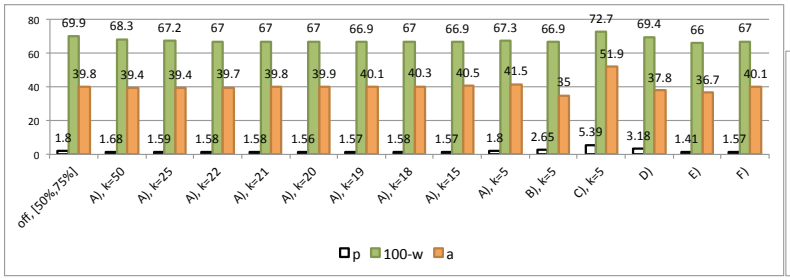


(a) Violations p, utilization $100 - w$, actions a

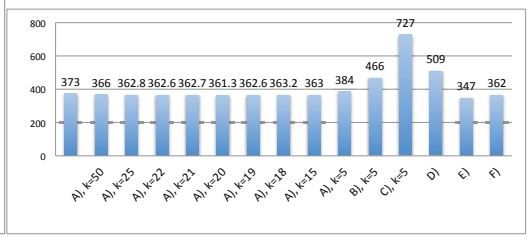


(b) Cost $c(p, w, a)$

Figure 2: Evaluation results for LOW workload volatility class



(a) Violations p, utilization $100 - w$, actions a



(b) Cost $c(p, w, a)$

Figure 3: Evaluation results for MEDIUM workload volatility class

workloads there is one case, where neither option outperforms the standard case, whereas in the other cases either option E or option F outperform the standard case.

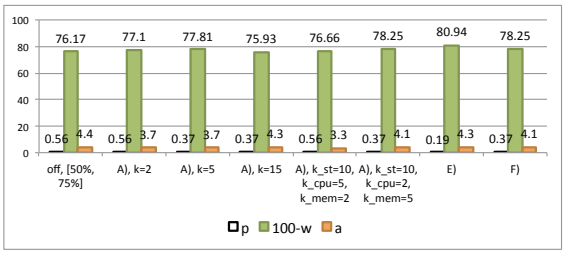
r_{min}		SLA parameter		r_{max}
1 GB	$\leq$	storage	$\leq$	1000 GB
1 Kbit/s	$\leq$	incoming bandwidth	$\leq$	80000 Kbit/s
1 Kbit/s	$\leq$	outgoing bandwidth	$\leq$	8000 Kbit/s
1 MIPS	$\leq$	CPU power	$\leq$	100000 MIPS
8 MB	$\leq$	memory	$\leq$	512 MB

Table IV: PovRay SLA

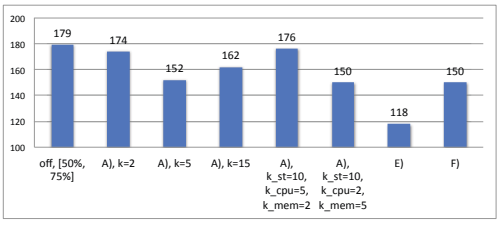
p	$100 - w$	a	$c(p, w, c)$	WV	AM	Details
5.56	63.8	17.0	754	POV_F1	off	[50%, 75%]
3.0	56.34	14.2	533	POV_F1	on	E)
3.0	53.44	11.7	544	POV_F1	on	F)
1.34	72.0	7.7	282	POV_F3	off	[50%, 75%]
1.12	71.7	7.8	261	POV_F3	on	E)
0.45	68.9	6.5	207	POV_F3	on	F)
3.24	72.9	15.8	475	POV_F9	off	[50%, 75%]
0.78	68.7	12.1	247	POV_F9	on	E)
1.34	70.1	18.4	302	POV_F9	on	F)
0.45	72.2	6.1	190	POV_B1	off	[50%, 75%]
0.44	73.0	6.0	186	POV_B1	on	E)
0.56	72.3	9.2	204	POV_B1	on	F)
0.11	71.2	10.1	161	POV_B2	off	[50%, 75%]
0.11	71.8	6.9	159	POV_B2	on	E)
0.22	72.5	10.8	171	POV_B2	on	F)
0.22	72.5	10.3	170	POV_B3	off	[50%, 75%]
0.45	71.8	8.8	194	POV_B3	on	E)
0.34	69.7	6.0	191	POV_B3	on	F)

Table V: Results for PovRay workloads

The SLA of the bionformatics workflow is defined as follows: 1 MB $\leq$ *storage* $\leq$ 19456 MB, 1 MIPS $\leq$ *CPU Power* $\leq$ 20000 MIPS, and 768 MB $\leq$ *memory* $\leq$ 8192 MB. Figure 5 reveals that all evaluated autonomic options outperform the standard case with option E achieving by far the best result. For option A we have also experimented with varying k for different resources and could achieve the second best result (tied with option F) by setting $k = 10$ for storage, $k = 2$ for CPU, and $k = 5$ for memory.



(a) Violations p, utilization $100 - w$, actions a



(b) Cost $c(p, w, a)$

Figure 5: Evaluation results for the bioinformatics workflow

Concluding we find that for 11 out of 14 real-world work-

loads both options E and F of the self-adaptive approach achieve better results than the static approach for at least 7% (workload POV_F2) and at most 48% (workload POV_F9). From the remaining workloads, for two of them (POV_B1 and POV_B2) only option E performs better, and for only one workload the static approach outperforms both self-adaptive ones by 11% (POV_B3).

VI. Conclusion

In this paper we have devised several methodologies for autonomically adapting parameters of a Cloud resource management framework on the level of VM reconfiguration. The goal of the approach is to reduce SLA violations, increase resource utilization and achieve both by a low number of reconfiguration actions.

We have devised two groups of strategies: the first one is based on a cost function that reflects the goal of the approach. The other strategy is based on classifying the workload into workload volatility classes. It acts according to this classification by either applying the substrategy of pre-configured parameters or the substrategy of applying the most appropriate strategy from the first group. In most cases we have seen that strategies from the latter group achieve better results for both substrategies, and outperform the strategies not taking workload volatility into account. Thus, we can deduce that workload volatility is an important aspect for governing Cloud infrastructures. Corresponding research is still at its beginning.

For future work we want to prove the benefit regarding the energy consumption of this approach. We will be able to not only capture the improvement in costs of the self-adaption, but also the reduction in energy consumption as compared to a non-self-adapting approach. Furthermore, we plan to investigate if we can generalize the findings for autonomically adapting approaches for other levels of governing Cloud infrastructures, e.g., VM migration or PM power management.

Acknowledgments

We want to thank Ivan Breskovic for his valuable comments on this work, which has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT08-018 and by COST-Action IC0804 on Energy Efficiency in Large Scale Distributed Systems.

References

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

[2] (FOSII) - Foundations of Self-governing ICT Infrastructures. http://www.infosys.tuwien.ac.at/linksites/FOSII (March 2012)

[3] Bahati, R.M., Bauer, M.A.: Adapting to run-time changes in policies driving autonomic management. In: ICAS '08: Proceedings of the 4th Int. Conf. on Autonomic and Autonomous Systems. IEEE Computer Society, Washington, DC, USA (2008)

[4] 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)

[5] Borgetto, D., Casanova, H., Costa, G.D., Pierson, J.M.: Energy-aware service allocation. Future Generation Computer Systems 28(5), 769 – 779 (2012)

[6] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599 – 616 (2009)

[7] Dutreilh, X., Rivierre, N., Moreau, A., Malenfant, J., Truck, I.: From data center resource allocation to control theory and back. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. pp. 410 –417 (July 2010)

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

[9] Emeakaroha, V.C., Netto, M.A.S., Calheiros, R.N., Brandic, I., Rose, C.A.F.D.: Desvi: An architecture for detecting SLA violations in cloud computing infrastructures. In: CloudComp 2010. Barcelona, Spain (October 2010)

[10] Hoyer, M., Schröder, K., Nebel, W.: 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. pp. 51–60. e-Energy '10, ACM, New York, NY, USA (2010)

[11] Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. 40(3), 1–28 (2008)

[12] Kalyvianaki, E., Charalambous, T., Hand, S.: Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In: Proceedings of the 6th international conference on Autonomic computing. pp. 117–126. ICAC '09, ACM, New York, NY, USA (2009)

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

[14] Koomey, J.G.: Worldwide electricity used in data centers. Environmental Research Letters 3(3), 034008 (2008)

[15] Łabaj, P.P., Leparc, G.G., Linggi, B.E., Markillie, L.M., Wiley, H.S., Kreil, D.P.: Characterization and improvement of RNA-Seq precision in quantitative transcript expression profiling. Bioinformatics 27(13), i383–i391 (2011)

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

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

[18] Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E., Pendarakis, D.: Efficient resource provisioning in compute clouds via VM multiplexing. In: Proceeding of the 7th international conference on Autonomic computing. pp. 11–20. ICAC '10, ACM, New York, NY, USA (2010)

[19] Padala, P., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K.: Adaptive control of virtualized resources in utility computing environments. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. pp. 289–302. EuroSys '07, ACM, New York, NY, USA (2007)

[20] Petrucci, V., Loques, O., Mossé, D.: A dynamic optimization model for power and performance management of virtualized clusters. In: e-Energy '10. pp. 225–233. ACM, New York, NY, USA (2010)

[21] Ramirez, A.J., Knoester, D.B., Cheng, B.H., McKinley, P.K.: Applying genetic algorithms to decision making in autonomic computing systems. In: Proceedings of the 6th international conference on Autonomic computing. pp. 97–106. ICAC '09, ACM, New York, NY, USA (2009)

[22] Rao, J., Bu, X., Xu, C.Z., Wang, L., Yin, G.: Vconf: a reinforcement learning approach to virtual machines auto-configuration. In: ICAC '09. pp. 137–146. ACM, New York, NY, USA (2009)

[23] Song, B., Hassan, M., nam Huh, E.: A novel heuristic-based task selection and allocation framework in dynamic collaborative cloud service platform. In: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. pp. 360 –367 (December 2010)

[24] Trapnell, C., Pachter, L., Salzberg, S.L.: Tophat: discovering splice junctions with RNA-Seq. Bioinformatics 25(9), 1105–1111 (2009)

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

[26] Yazir, Y., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., Coady, Y.: 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 (July 2010)