Optimization Method for Request Admission Control to Guarantee Performance Isolation

Rouven Krebs
SAP AG
Applied Research
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
Rouven.Krebs@sap.com

Philipp Schneider
SAP AG
Applied Research
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
Philipp.Schneider01@sap.com

Nikolas Herbst
FZI Research Center for
Information Technology
Haid-und-Neu-Strasse 10-14
76131 Karlsruhe, Germany
herbst@fzi.de

ABSTRACT

Software-as-a-Service (SaaS) often shares one single application instance among different tenants to reduce costs. However, sharing potentially leads to undesired influence from one tenant onto the performance observed by the others. Furthermore, providing one tenant additional resources to support its increasing demands without increasing the performance of tenants who do not pay for it is a major challenge. The application intentionally does not manage hardware resources, and the OS is not aware of application level entities like tenants. Thus, it is difficult to control the performance of different tenants to keep them isolated. These problems gain importance as performance is one of the major obstacles for cloud customers. Existing work applies request based admission control mechanisms like a weighted round robin with an individual queue for each tenant to control the share guaranteed for a tenant. However, the computation of the concrete weights for such an admission control is still challenging. In this paper, we present a fitness function and optimization approach reflecting various requirements from this field to compute proper weights with the goal to ensure an isolated performance as foundation to scale on a tenants basis.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Performance attributes, Modeling techniques*

Keywords

SaaS, Multi-Tenancy, Performance, Isolation, Scalability

1. INTRODUCTION

Cloud Computing increases in importance, because it offers a huge potential for cost saving [14]. The NIST differentiates between three general service models. One is Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) [11].

A SaaS application is hosted centrally and customers access it via a thin client such as a web browser. The complete infrastructure, like storage, is managed by the application provider. Examples for SaaS are web based email services, Google Apps and SAP Business ByDesign. Sharing of resources promises high cost savings by reducing one time costs and increasing utilisation of the resources. Since the highest level of sharing occurs on the application layer, the highest potential of cost savings lies within the SaaS level, where several users share a single application instance. Multi Tenancy describes the approach where tenants, are served and charged as an individual entity while running on a single application instance with other tenants. A tenant, typically a legal body, is defined as a group of users, which share the same view onto an application. However, this comes along with additional challenges like the isolation of the tenants. In multi-tenant applications (MTA), it is the application providers task to isolate the data, configuration and the performance observed by the tenants. While the isolation of data is primary an engineering task the isolated control of performance is still an open research topic and a reliable performance is one of the major obstacles for potential cloud customers [2].

In multi-tenant scenarios typically service levels have been agreed with tenants in advance and oblige the provider to maintain a certain performance level for each tenant. This is denoted with the term *Service Level Agreements* (SLAs) which is specified individually for each tenant. The quality depends mainly on its willingness to pay for a certain QoS. In order to work with SLAs, performance needs to be measured. This is done using metrics like response time as the guarantee to be achieved by the system as long as a certain quota like the request arrival rate or amount of active users is not exceeded by the tenant. A tenant's willingness to pay for service performance guarantees may change over time e.g., if the business goes well it might result in an increasing demand onto the systems. Therefore, a given quota might not be able to fulfil the demands of a tenant. While the tenant starts a renegotiation with the provider to increase its quota the provider must be able to increase the share a tenant is allowed to consume. Using elasticity/scalability features of the application by adding additional resources does not solve the problem, because the additional resources would be shared among all tenants using the system. Consequently, a mechanism is needed to assure that new resources

HotTopiCS 2014 Dublin, Ireland

will only be used by the tenant who pays for them.

Thus, there is a need to isolate the performance different tenants observe. In [7], performance isolation is defined as follows: "A system is performance isolated, if for [tenants] working within their quotas the performance is not affected [even if] other [tenants] exceed their quotas". By changing the quota for a tenant, we can ensure that the amount of resources they can use when the system is under high load correspond to their SLA defined quota and consequently maintain the guaranteed response time.

One approach to control the performance on a tenant's basis is to leverage a request based admission control. The requests originating from different tenants have to pass a Request Admission Control (RAC). The RAC forwards, delays, or even rejects requests based on the tenant it is originating from to make sure a tenant does not exceed its quota in situations it would influence others. Our system uses a modified version of a weighted fair queueing scheduler, whereby every tenant has its separate queue. Our RAC ensures, that for a given period of time the amount of admitted requests per tenant corresponds to their weight, even in situations where one tenant starts to send its requests late in a period. This is ensured by a dynamic adoption of the weights within a period. However, the reference values i.e. weights for each period have to be derived by the SLAs. The controller's task is to derive these reference values.

Several techniques for a RAC were already published (e.g., [7, 9, 10]). However, finding a suitable mechanism to find good reference weights for the tenants used by the RAC is still challenging. This problem is similar to mapping high level SLAs to low level resource guarantees which is still an open research question [3].

In this paper, we present a fitness function and optimization approach with two configuration parameters based on an analytical system model of the RAC and server to derive suitable weights for the weighted fair queueing scheduler. The proposed solution is independent of the concrete system function which is individual for each application environment. The mechanism could be applied in the controller of the aforementioned system. The concrete realization of the system model and the RAC are not in the scope of this paper due to space limitations.

The reminder of the paper is structured as follows. In Section 2, we conduct related work. Section 3 presents how to derive the weights by a given SLA with the fitness function and optimization developed. The following Section 4 discusses how various configuration parameters influence the behaviour of the optimization and Section 5 concludes the paper.

2. RELATED WORK

In the following, an overview of the most relevant literature about performance isolation and multi-tenancy is presented.

Zhang [16] and Fehling [4] are representatives of approaches where isolation is tried to be achieved by placing tenants onto compute nodes in a way they do not influence each other but maximizing the resource usage to increase efficiency. The placement is based on the estimated resource demands for a particular tenant as required to fulfil the SLAs. These approaches are similar to traditional Bin Packing problems which are proofed to be NP-hard and therefore heuristics to optimize the placement are necessary. Furthermore, a good estimation and forecast of the required resources is needed which is difficult in cloud environments and there is still an influence of tenants co-located on the same node. In comparison the approach in this paper is based on request admission control and derive an analytical model from the application which could be numerically optimized.

Control theory in general [6] requires a time behaviour analysis and models of the system to be controlled. This information is used to dynamically optimize the system which is prone to need some time and comes along with the risk of oscillating configurations. The most work in this area does not consider multi-tenancy or admission control with numerous configuration parameters like the weights for each tenant. Furthermore, we do not need a dynamic adaptation because the optimization is done on a response time model and thus we avoid the mentioned drawbacks.

[10] is an example of an implementation using a two-levelled approach based on control theory. On a higher level (L1) a referential average response time is regulated by controlling the admission rates of requests of individual tenants. On an inner level (L0) a second controller, which is responsible for service differentiation, i.e. performance isolation, is employed. It enforces the SLAs of each tenant by controlling the priorities to regulate the respective response times. In order to do that, simplifying assumptions like linear correlation between response time and priority are made. Although overcommitted systems are prevented by the L1-controller, the simplification used for the L0 controller probably fails to describe an overloaded system, due to the highly non-linear behaviour of queuing systems approaching maximum utilisation. Furthermore, queuing in front of the L0 controller is not possible, therefore the system will fail to deliver the service although the response times including queuing would be sufficient. The SPIN environment [9] interprets the ratio of the mean arrival rate and mean service rate to detect performance anomalies/overload of the system. In a second step the aggressive tenant, that consumes the most resources is identified and its request flow is adopted. This approach concludes from the computed ratio the observed performance, whereas our approach is directly based on the SLAs.

In our former work [7], we introduced static request admission mechanisms like Round Robin or Black Lists. However, with static weights these mechanisms cannot react to changing requirements. Furthermore, a service differentiation between different tenants is not possible because the optimization to find an optimal configuration for the admission control was not discussed.

Other approaches like [15] and [8] use statistical analysis or fine grained measurements to estimate the resources used by a tenant. Based on this knowledge they control the request flow/admission to prevent one tenant from allocating more resource then he is allowed by the quota. Thus these mechanisms isolate resources. However, they do not reflect the SLA relevant response times. Furthermore, the estimation of resource demand is challenging and a fine grained measurement is often not possible.

Gupta et al. [5] is one of the authors developing mechanisms to isolate VMs. Gupta determines the CPU overhead for I/O of a XEN Domain onto Dom0 and adjusts the CPU scheduler based on this to achieve lower inference. However, all the approaches isolating virtual machines can directly access the hardware and schedulers which is not possible in

our case.

3. DERIVATION OF WEIGHTS

In this section we develop a fitness function $f : \mathbb{R}_+^n \to \mathbb{R}$ that measures the degree by which the performance is isolated, depending on the weight vector $\mathbf{w}$ of weights assigned to tenants and a system function R_i describing the response time of the system for tenant i based in the tenants weight w_i.

In addition to w_i, given through the weight vector $\mathbf{w}$, the system performance function R_i might depend on system parameters like the number of users m_i per tenant, amount of work per request and think times which cannot actively be controlled but influence the optimal weights. Thus these parameters are assumed to be constant of the current control interval. Thus, we are optimizing the weight vector $\mathbf{w}$ in a way every tenant achieves the optimal resource time according to the fitness function introduced later. The sum of all $w_i = 1$. As a precondition for our optimisation R_i has to be convex in $[0, 1]$ and unbounded for $w \to 0$.

The fitness function is designed such, that it is low in value for weight vectors $\mathbf{w}$ that isolate performance well, and increasingly higher in value for tenant weights $\mathbf{w}$ that are increasingly unsuited to isolate performance. This way, by minimising f, the weight vector which isolates performance best, can be determined for a given scenario.

In the following, the fitness function f is constructed in such a way, that it reflect the definition of performance isolation. This way, it can be handled analytically convenient and can be used in different scenarios with varying system performance functions. The fitness function f also allows adjust-ability, by means of tuning parameters in order to allow further service differentiation where one tenant may be allowed to consume more resource.

3.1 Optimization Problem Statement

Let $M := \{\mathbf{w} = (w_1, \ldots, w_n)^T \in \mathbb{R}_+^n \mid \sum_{i=j}^n w_j = 1\}$ be the space of normed weight vectors. E.g. a weight vector $\mathbf{w} \in M$ denotes potential weights assigned to tenants.

Note that for each resource distribution mechanism, that relies on non-negative numerical values $\tilde{\mathbf{w}} \in \mathbb{R}_+^n$ to assign portions of resources to tenants, these values $\tilde{\mathbf{w}}$ can be mapped to M via normalisation, which is an affine transformation.

As outlined in the introduction performance guarantees have to reflect tenants quotas. Let therefore g_i be the guaranteed response time of t_i and let q_i be its quota.

As a first component of the fitness function f, a *violation function* is defined, which measures the degree by which the performance guarantees g_i of a tenant t_i is violated. This function should be high in value if SLAs are violated that is if $R_i(w_i) > g_i$ and it should be near zero if not, that is if $R(w_i) \leq g_i$. For this purpose the function $v_i : \mathbb{R}_+ \to \mathbb{R}_+$, that maps weights to the degree of guarantee violation

$$v_i(w_i) := \exp\left(c_v \frac{R_i(w_i) - g_i}{g_i}\right), \tag{1}$$

is defined, which has the aspired properties. The degree of the guarantee violation of a tenant is measured relative to its guarantee g_i. It is relatively small (≤ 1) if the performance guarantee of t_i is not breached and high if that is the case. The exponential trend enforces exponentially increasing penalties for graver violations of guarantees of tenants, thus starkly prioritising the improvement of their performance over the improvement of the violations of tenants, that do no violate their guarantees so gravely.

The constant c_v is a tuning parameter and determines how strong a performance guarantee violation is weighted. For high values of c_v response time violations are stronger penalised than for lower values. The constant c_v can also be made tenant specific (c_v^i), thus prioritising the fulfilment of performance guarantees of tenants with higher values of c_v^i, over those with lower values. This way an additional distinction between customers, according to their payment level can be achieved (e.g. gold-, silver- or bronze-customers).

The quota describes either the number of users of a tenant or the number of users multiplied by the average amount of work per user of a tenant. Therefore, let l_i be the workload of t_i. Note, that the exact definition of l_i is irrelevant for the subsequent derivations and therefore other definitions for the workload are applicable.

For the purpose of differentiation of tenants by their disruptiveness, a *penalty function* is defined, which penalises disruptive tenants, giving the fulfilment of their performance guarantees a lower priority as compared to abiding tenants. The penalty function assigns weights to tenants, whereas a low weight corresponds to a high penalty. A tenant is penalized more, if that tenant is disruptive, that is if $l_i > q_i$, than when being abiding, that is if $l_i \leq q_i$. On the basis of this criteria, the following penalty function $p : \mathbb{R}_+ \to \mathbb{R}_+$, maps a tenants workload l_i to its penalty

$$p_i(l_i) := (1 + \exp\left(c_p \frac{l_i - q_i}{q_i}\right))^{-1}. \tag{2}$$

This function is a flipped sigmoid function and will be used in the following to weaken $v_i(w_i)$, if t_i's workload exceeds its quota. The function p_i assumes values in $(0, 1)$ and has the asymptotes 0 for $l_i \to \infty$ and 1 for $l_i \to -\infty$. Its value is $1/2$ for $l_i = q_i$, thus q_i marks the point where tenant i becomes increasingly insignificant to the optimisation with further increasing workload l_i.

The constant c_p determines how abrupt the transition from 1 to 0 is. For high values of c_p the transition is more abrupt. Lower values of c_p stretch the transition, thus implicating a more gradual transition from one to zero than higher values, whereby quota breaches have less impact. Thus c_p should be used to adjust the rigidity of quota enforcement.

The constant c_p could again be made tenant specific (c_p^i) and would thus become a tuning parameter to distinguish tenants by their service level.

Note that $p_i(l_i)$ is independent of the weight vector $\mathbf{w}$. Since l_i is an operating parameter which is assumed to be predetermined an thus fixed, $p_i(l_i)$ is constant and thus simply used in the form p_i in the following. For this reason the following results will also remain unaffected if a different definition of p_i is used.

In case of a RAC described in the Introduction the optimisation would go the easiest way, and first increase the weights of lighter tenants with smaller amounts of users, as this results in the biggest return in response time and thus fitness. This effect has to be reduced. An additional factor is required to compensate this. Therefore, let

$$h_i := \frac{l_i \cdot n}{\sum_{j=1}^n l_j}$$

be the *heaviness* of t_i. The heaviness describes how big the workload l_i of t_i is, in relation to the average workload $(\sum_{j=1}^{n} l_j)/n$ of all tenants. A tenant t_j is said to be *heavier*, than a tenant t_k if $h_j > h_k$.

Building on the previously defined terms, the following *tenant specific fitness function* $f_i := [0,1] \to \mathbb{R}_+$ for t_i is defined as

$$f_i(w_i) := h_i \cdot p_i \cdot v_i(w_i). \tag{3}$$

In $f_i(w_i)$ the violation function $v_i(w_i)$ is normalised by the penalty p_i and the heaviness h_i of t_i. The performance of a heavy tenant t_j is more expensive to improve than that of a lighter tenant t_k. This means, that under otherwise equal conditions, and increase of the heavier tenant t_j's weight w_j by a certain amount does not improve response time as much as compared to a the lighter tenant t_k. Thus, if the factor h_i were omitted, the optimisation would go the easiest way, and first increase the weights of lighter tenants, as this results in the biggest return in response time and thus fitness.

The following lemma shows that the described effect of heaviness of tenants on the optimisation is avoided through normalisation of the f_i by h_i.

LEMMA 1. *Let t_j and t_k be two tenants in a Multi Tenancy system, whose performance is given by differentiable functions $R_j, R_k \in \mathcal{C}^1(0,\infty)$, which are equal when dilated by their respective workloads, i.e.*

$$R_j(l_j w) = R_k(l_k w),\ w \in (0,\infty).$$

Further let their respective utilisation of quotas q_j, q_k be equal and thus their penalties $p_j = p_k$. Additionally let their respective guarantees be equal $g_j = g_k$. Then the derivatives of the respective fitness functions f_j and f_k are equal, for all weights, for which the respective performance functions R_j, R_k are equal, that is for all $w \in (0,\infty)$

$$f_j'(l_j w) = f_k'(l_k w),$$

where $f_j' := \frac{d}{dw} f_j$ and $f_k' := \frac{d}{dw} f_k$ are the derivatives with respect to w of the according fitness functions.

PROOF.

$$\begin{aligned}
f_j'(w) &= \frac{d}{dw}\, h_j \cdot p_j \cdot v_j(w) \\
&= h_j \cdot p_j \cdot \frac{d}{dw} \exp\left(c_v\, \frac{R_j(w) - g_j}{g_j}\right) \\
&= h_j \cdot p_k \cdot \frac{d}{dw} \exp\left(c_v\, \frac{R_k(w\, l_k/l_j) - g_k}{g_k}\right) \\
&= h_j \cdot p_k \cdot \frac{d}{dw} v_k(w\, l_k/l_j) \\
&= \frac{l_j \cdot n}{\sum_{i=1}^{n} l_i} \cdot \frac{l_k}{l_j} \cdot p_k \cdot v_k'(w\, l_k/l_j) \\
&= h_k \cdot p_k \cdot v_k'(w\, l_k/l_j) = f_j'(w\, l_k/l_j).
\end{aligned}$$

The substitution $\hat{w} := w/l_j$ concludes the proof. □

Note that the performance functions R_j, R_k must be congruent when dilated for their respective workloads l_j, l_k is usually met for performance functions modelling real systems. This is due to the fact, that the input weights w_j, w_k are normalised by their respective workloads l_j, l_k in the according performance functions R_j, R_k.

Lemma 1 reveals the effect of h_i on the fitness function f_i.

By introducing the factor h_i into (3), the gain of changes in t_j's and t_k's weights w_j, w_k is equal, when they have equal parameters despite their heaviness. That implies, that minimizing $f_j(w_j) + f_k(w_k)$ must result weights w_j, w_k such that the respective response times R_j, R_k are equal. This way the effect, that tenants are disadvantaged due to their heaviness, is circumvented.

Composing (3) into a global fitness function, the following optimisation problem is obtained

$$\begin{aligned}
&\text{min!} \quad f(\mathbf{w}) := \sum_{j=1}^{n} f_j(w_j), \\
&\text{subject to} \\
&\mathbf{w} \in M \iff \begin{cases} \sum_{j=1}^{n} w_j = 1, \\ w_j \geq 0, \quad \forall j \in \{1,\dots,n\}. \end{cases}
\end{aligned} \tag{4}$$

3.2 Problem Analysis

In this section the tenant specific fitness functions $f_i(w_i)$ of t_i given in (3) will be further analysed. This way new perspectives of the parameters influencing f_i are provided.

The function $f_i(w_i)$ is composed of the violation function $v_i(w_i)$ given in (1), the penalty p_i given in (2) and the heaviness factor h_i. The following transformations reveal, how the violation function $v_i(w_i)$ and the values p_i, h_i interact.

$$\begin{aligned}
f_i(w_i) &= h_i \cdot p_i \cdot v_i(w_i) \\
&= h_i \cdot p_i \cdot \exp\left(c_v\, \frac{R_i(w_i) - g_i}{g_i}\right) \\
&= h_i \cdot \exp\left(c_v\, \frac{R_i(w_i) - g_i}{g_i} + \log(p_i)\right) \\
&= h_i \cdot \exp\left(c_v\, \frac{R_i(w_i) - g_i(1 - \log(p_i)/c_v)}{g_i}\right)
\end{aligned}$$

Since the definition of (2) implies $p_i < 1$, it follows that $\log(p_i) < 0$. Thus the term $(1 - \log(p_i)/c_v)$ in the formula above is strictly bigger than 1. This means, that the performance guarantee of t_i is shifted by a factor of $(1 - \log(p_i)/c_v) > 1$ to the right, hence shifting the point, where the response time causes the violation function to become noticeably big (≥ 1) further to the right.

If t_i is abiding, then p_i is near 1 (see (2)), which means that $\log(p_i) \approx \log(1) = 0$. Therefore $(1 - \log(p_i)/c_v) \approx 1$ and hence the shift of g_i to the right is hardly noticeable. Thus for abiding tenants performance isolation functions as usual. However if t_i becomes increasingly disruptive, then p_i moves even closer to 0 and hence $(1 - \log(p_i)/c_v)$ tends to ∞ quickly.

This means, that the more disruptive t_i gets, the farther its guarantee is shifted to the right, therefore "allowing" the optimisation to assign t_i a performance via its weight w_i, which is beyond its guaranteed performance g_i, without major increases of the violation function (1). Therefore t_i is treated as if its performance guarantee were fulfilled, even though the actual performance of t_i may be above g_i. Only when the performance of t_i becomes so bad, i.e. the response time become so large, that it increases beyond the shifted guarantee $g_i(1 - \log(p_i)/c_v)$, the violation function starts to kick back in, becoming ≥ 1.

Similarly as above the following transformation is obtained

$$f_i(w_i) = p_i \cdot \exp\left(c_v \, \frac{R_i(w_i) - g_i(1 - \log(h_i)/c_v)}{g_i}\right).$$

From this transformation it becomes evident, that h_i has an effect similar to that of p_i. However, since h_i is the relative portion of heaviness of t_i, compared to the average heaviness, it is valued in $[0, \infty)$. If for example if t_i is twice as "heavy" as the average heaviness, then $h_i = 2$. If on the other hand the tenant is half as heavy, then $h_i = 0.5$.

If a tenants heaviness is $h_i = 1$ then $\log(h_i) = 0$ and thus h_i has no effect. If $h_i > 1$, i.e. t_i is heavier than the average, then $(1 - \log(h_i)/c_v) < 1$ which means that t_i's guarantee is decreased, thus improving it. If on the other hand $h_i < 1$, then $(1 - \log(h_i)/c_v) > 1$, thus increasing t_i's guarantee, in other words downgrading it. For the special case that a tenant has no workload, i.e. $l_i = 0$ then $h_i = 0$ as well, implicating $f_i \equiv 0$ as can be seen from equation (3).

Thus h_i has a similar effect on the displacement of guarantees as p_i, with the difference that a tenants guarantee can even be improved by decreasing it. The reason why the latter is done, is to compensate the effect of a tenants heaviness, as it was argued and proofed in lemma 1. Finally (3) can be written in the following form

$$f_i(w_i) = \exp\left(c_v \, \frac{R_i(w_i) - g_i(1 - \log(h_i)/c_v - \log(p_i)/c_v)}{g_i}\right).$$

Evidently the effects of shifting of t_i's guarantees g_i caused by p_i and h_i, may strengthen each other or may cancel each other out.

Following up on the equation above, the function $f_i(w_i)$ can be shortened to

$$f_i(w_i) \;=\; \exp\left(\alpha_i R_i(w_i) - \beta_i\right), \tag{5}$$

with $\alpha_i := c_v/g_i$ and $\beta_i := c_v(1 - \log(h_i)/c_v - \log(p_i)/c_v)$, whereas $\alpha_i, \beta_i > 0$. Furthermore the argument of the exponential function is subsumed into one variable, termed *violation value* $V_i(w_i)$ of t_i which is thus given by

$$V_i(w_i) \;:=\; \alpha_i R_i(w_i) - \beta_i.$$

The violation value $V_i(w_i)$ determines to what extent the SLAs of t_i are violated, in consideration of the heaviness h_i of t_i. This means that the violation value $V_i(w_i)$ determines the size of $f_i(w_i) = \exp(V(w_i))$. If $V_i(w_i) \leq 0$ then $f_i(w_i) = \exp(V(w_i))$ is relatively small (≤ 1). If on the other hand $V_i(w_i) > 0$, then t_i's performance guarantees and so its SLAs are violated and therefore $f_i(w_i) = \exp(V(w_i))$ is relatively large (> 1) and also fast increasing for even greater violation values.

If $V_i(w_i) > 0$ for a tenant t_i, this means, that this tenant is not assigned enough capacity κ_i in order to satisfy the corresponding SLAs. Otherwise if $V_i(w_i) < 0$ the capacity κ_i of t_i can be reduced and still t_i's SLAs are met. The capacity κ_i assigned to t_i should thus be dimensioned such that $V_i(w_i) = 0$, in order not to violate t_i's SLAs and not to waste resources by over-fulfilling them.

Thus the violation values $V_j(w_j)$, $j \in \{1, \ldots, n\}$ indicate, how adequate the dimensioning of system capacity κ is for the handling of the given set of tenants together with their SLAs. If $\sum_{j=1}^{n} V_j(w_j) \ll 0$ then system capacity κ can be dimensioned smaller, the number of tenants can be increased or the SLAs can be defined in tighter bounds. If however $\sum_{j=1}^{n} V_j(w_j) \gg 0$, then either κ should be increased, the number of tenants be decreased, or their SLAs be defined within looser bounds.

3.3 Relevant Optimization Characteristics

In the following, we show that the underlying fitness function of the optimisation problem (4) is convex, and that the unique minimum in M lies in the interior of M which allows to solve the optimisation problem relatively convenient using a numerical method. For this purpose, the following theorem is proposed.

THEOREM 1. *If the fitness function $f(\mathbf{w})$, given through the optimisation problem 4 is combined with performance functions $R_j : (0, 1] \to \mathbb{R}_+$, $j \in \{1, \ldots, n\}$ that are convex on $[0, 1]$, then $f(\mathbf{w})$ is convex for $\mathbf{w} \in M$ as well. Additionally, the unique solution $\hat{\mathbf{w}}$ of the optimisation problem 4 lies in the interior of M if $\lim_{w \to 0} R_j(w) = \infty$.*

PROOF. First it is proofed, that $f(\mathbf{w})$ is convex. For this purpose it is shown, that for each tenant t_i the tenant specific fitness functions $f_i(w_i)$ are convex. Consider $f_i(w_i)$ in the form of equation 5. Since, according to the premise, $R_i(w_i)$ is convex $\alpha_i R_i(w_i) - \beta_i$ is convex, as well.

Since for each convex, function $\varphi : \mathbb{R} \to \mathbb{R}$ the composition $\exp \circ \varphi$ is convex,

$$f_i(w_i) \;=\; \exp\left(\alpha_i R_i(w_i) - \beta_i\right)$$

is convex as well. It is easily shown, that the sum $h(x_1, \ldots, x_n) = \varphi_1(x_1) + \ldots + \varphi_n(x_n)$ of convex functions $\varphi_1, \ldots, \varphi_n : \mathbb{R} \to \mathbb{R}$ is again convex. Hence the global fitness function

$$f(\mathbf{w}) \;=\; \sum_{j=1}^{n} f_j(w_j)$$

is convex. Since M is a compact, the extreme value theorem applies, which implies that there exists a weight vector $\hat{\mathbf{w}}$ for which f attains its minimum. Due to the fact that f is convex a local minimum must also be the unique, global one.

It remains to be shown, that the unique minimum $\hat{\mathbf{w}}$ lies in the interior $\mathring{M}$ of M. Suppose, for contradiction, that the minimum lies on the boundary of M, i.e. $\hat{\mathbf{w}} \in \partial M$. As each weight vector in ∂M has a component which is zero, it holds that for $\hat{\mathbf{w}} = (\hat{w_1}, \ldots, \hat{w_n}) \in \partial M$ there exists an $i \in \{1, \ldots, n\}$, such that $\hat{w_i} = 0$. Since $\lim_{w \to 0} f_j(w) = \infty$ it holds that

$$f_i(\hat{w_i}) \;=\; \infty \quad \Longrightarrow \quad f(\hat{\mathbf{w}}) \;=\; \infty.$$

However for each $\mathbf{w} = (w_1, \ldots, w_n) \in \mathring{M}$, it holds that $w_j > 0 \; \forall j \in \{1, \ldots, n\}$, thus

$$f_j(w_j) < \infty \quad \forall\, j \in \{1, \ldots, n\} \quad \Longrightarrow \quad f(\mathbf{w}) < \infty \;=\; f(\hat{\mathbf{w}}).$$

This is a contradiction to the statement, that $\hat{\mathbf{w}}$ is the minimum. Hence $\hat{\mathbf{w}} \in \mathring{M}$. □

Consequently, if the fitness function $f(\mathbf{w})$, given through the optimisation problem 4 is combined with a valid response time function R_j, then $f(\mathbf{w})$ is convex for $\mathbf{w} \in M$ and the unique minimum $\hat{\mathbf{w}}$ lies in the interior $\mathring{M}$.

If a concrete optimisation method which converges to a local minimum (like e.g. gradient descend) is used, together with a starting point $\mathbf{w}_{\text{init}} \in \mathring{M}$ it will converge to the local

minimum in the interior $\mathring{M}$ of M, since $f(\mathbf{w}) = \infty$ if $w_i \to 0$ for any component w_i of $\mathbf{w}$. Thus an optimisation method converging to a local minimum must converge to $\hat{\mathbf{w}}$, even without the additional constraints.

It is worth to mention, in practice an optimisation method may jump outside of M, due to a step length, or due to an optimisation method, which does not converge to the nearest minimum, but attempts to optimise globally like e.g. a genetic optimisation algorithm. Thus, depending on the optimisation method used, it may still be necessary to constrain the feasible set of weights as follows

$$w_1, \ldots, w_{n-1} \geq \varepsilon, \quad w_l + \ldots + w_{n-1} \leq 1 - \varepsilon,$$

for a small $\varepsilon > 0$ close to 0. The threshold of ε proofed to be necessary in some cases, since some optimisers allow for a certain numeric tolerance in the constraints. If the threshold of ε were omitted and one of the constraints were violated only by a very small value $\delta > 0$ close to 0, then this would imply $w_j = -\delta < 0$ for one $j \in \{1, \ldots, n\}$ and thus the system function R might return unrealistic values which in fact result in wrong optimizations.

4. OPERATIONAL BEHAVIOUR AND PARAMETERS

In this section it will be determined, how changes in operational parameters (number of users m_i), impacts weight distributions and performance. It will be shown how the tuning parameters c_v and c_p influence the performance behaviour which helps to find a suitable configuration for them.

For the optimisation, a computer algebra system [12] is used. The response times presented are based on the system performance function R which always had an steady-state error below 20% for a test environment running an enhanced version of the TPC-W Benchmark [1] on the application runtime environment of the SAP Hana Cloud [13]. Consequently, the results are representative for a real system.

Note that the plots given depend on operational parameters of the system (e.g., amount of users). Therefore, the characteristic plots used in this section to obtain good values for c_v and c_p might be different in other scenarios. However, the general observations will be the same.

4.1 Influence of Varying Workload

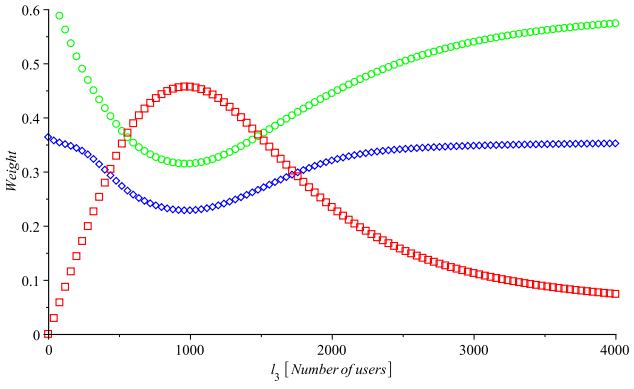

Figure 1: Weight With Increasing Load

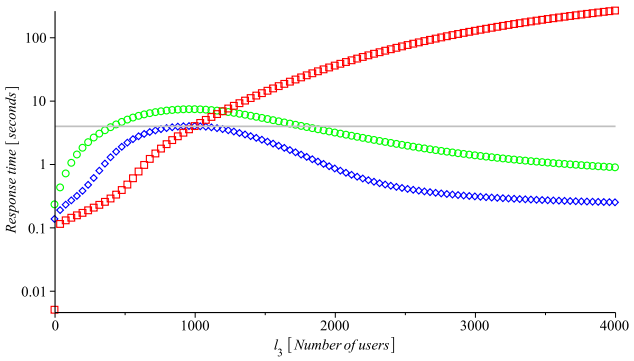

Figure 2: Response Time With Increasing Load

In Figure 1 a scatter plot of the weights and in Figure 2 the according response times are presented for an increasing load. The blue diamonds marks an abiding tenant t_1, whereat the green circles mark a mildly disruptive tenant t_2. The red boxes mark a tenant t_3 that is increasing its workload from $l_3 = 0$ to $l_3 = 4000$ users. All other parameters are kept fixed as follows $m_1 = q_1 = 500$, $m_2 = 900, q_2 = 750$, $q_3 = 1000$, seconds, $g_1, g_2, g_3 = 4$ seconds $c_v = 0.1$, $c_p = 5$.

Each point in the scatter plots represents the result of optimisation for a different workload of t_3. The plot shows how its changing penalty p_3 and heaviness h_3 influence the weight vector $\mathbf{w}$.

As depicted in Figure 1, t_3 starts with a workload of 0 users, for which the respective heaviness h_3 is 0 as well. Thus the fitness function is $f_3 \equiv 0$, which is why t_3 is disregarded by the optimisation method, hence given the weight $w_3 = 0$. However, for an increasing workload t_3 its assigned weight w_3 strongly increase due to the increasing heaviness h_3. The additional weight of t_3 is drawn from other tenants, therefore their weights are decreasing.

The increase in w_3 continues until l_3 approaches the quota q_3, at which the effect of the penalty p_3 becomes apparent. That means, that p_3 gradually converges to 0, thus making t_3 increasingly unimportant to the optimisation. The farther the workload of t_3 surpasses its assigned quota q_3, the more severe its penalty gets, and the farther w_3 deteriorates. Note that the heaviness h_3 grows linearly in l_3, whereas the penalty p_3 declines hyperbolic exponentially, i.e. asymptotically in $\mathcal{O}\left(\frac{1}{\exp(l_3)}\right)$.

That is an important feature, since it implies, that the positive effect (from t_3's point of view) of increasing heaviness h_3, is at some point eclipsed by the counteractive effect of the penalty p_3 which declines much faster, i.e. $p_3 \cdot h_3 \to 0$ for $l_3 \to \infty$. Consequently, around $l_3 = q_3$, the weight w_3 of t_3 approaches 0, since it t_3's fitness f_3 becomes increasingly unimportant to the overall fitness f. This way it is ensured, that a tenant cannot increase the amount of resources assigned to it indefinitely, by simply increasing its workload.

When t_3's workload reaches its quota, the two abiding tenants t_1, t_3 have a lower response time then the medium disruptive tenant t_2 (cf. Figure 2). When t_3 gets disruptive, by surpassing its quota, the respective response time rises significantly over the response time of the other tenants, while theirs even decreases below their quotas, since

resources of t_3 become free.

4.2 Influence of Violation Factor

Figure 3 depicts the weights and Figure 4 the according response times determined via optimisation for different values of the parameter c_v on the x-axis. The value of the parameter c_v is varied in the range from 0.001 to 10. All other parameters are kept fixed as follows $m_1 = q_1 = 500$, $m_2 = 900, q_2 = 750$, $m_3 = 1500, q_3 = 1000$ seconds, $g_1, g_2, g_3 = 4$ seconds, $c_p = 5$.

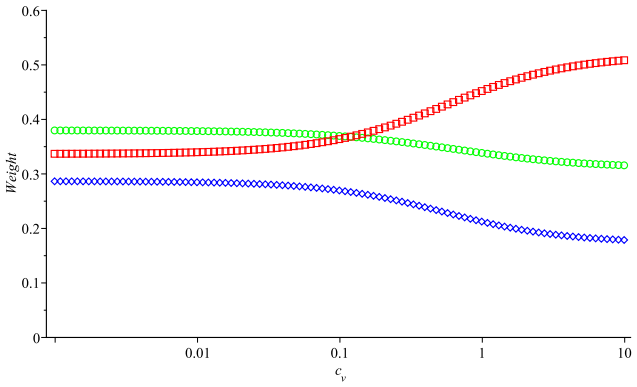

Figure 3: Weights With Increasing c_v

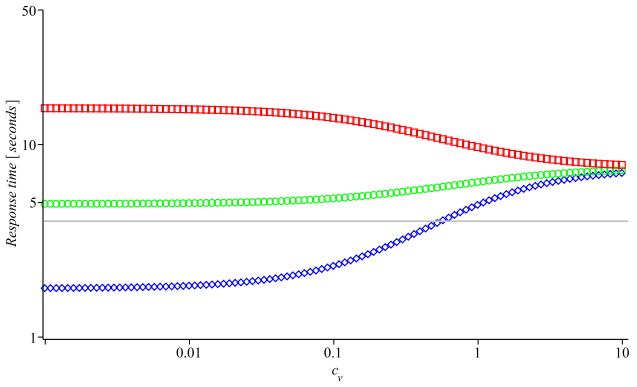

Figure 4: Response Time With Increasing c_v

Thus the configuration is similar to the previous example with changing values of c_v instead of a flexible workload. The weight distribution of tenants t_j converges to values that are proportional to their respective coefficients $h_j \cdot p_j$. This is due to the fact, that the derivatives of the according violation functions $\frac{\partial}{\partial w_j} v_j(w_j)$ given in (1) converge to 1, since

$$\lim_{x \to 0} \frac{\partial}{\partial x} \exp(x) = \lim_{x \to 0} \exp(x) = 1.$$

This means, that an increase in weight is equally rewarding in terms of fitness, for all tenants for values of c_v near to 0, thus making it only dependent on the proportion respective factors $h_j \cdot p_j$.

In contrast to that, it can be observed, that for high values of c_v the factor $h_j \cdot p_j$, is becoming increasingly insignificant. The reason for that is, that the derivative of the respective violation functions $\frac{\partial}{\partial w_j} v_j$ is becoming so large, that even small differences in relative guarantee violations of tenants become so tremendous in terms of those tenants fitness, that it cannot be compensated by $p_j \cdot h_j$. Thus, differences in guarantee violations are avoided by the optimiser, and hence the weights converge to the same response time for each tenant.

Summarized, one should avoid the extremes for very low or very high values of c_v. Thus a value for c_v in the plots given in Figure 3 and Figure 4 is chosen, where the effects of the isolation via $h_j \cdot p_j$ and relative guarantee violation are equal, therefore allowing for performance isolation the way it was intended. In this specific scenario, a value between 0.1 and 1 would be a suitable choice for c_v.

4.3 Influence of Penalty Factor

Figure 5 plots the weights and Figure 6 the according response times for different values of the parameter c_p. Parameter c_p is varied in the range from 0.1 to 100. All other parameters are maintained constant: $m_1 = q_1 = 500$, $m_2 = 900, q_2 = 750$, $m_3 = 1500, q_3 = 1000$ seconds, $g_1, g_2, g_3 = 4$ seconds, $c_v = 0.1$.

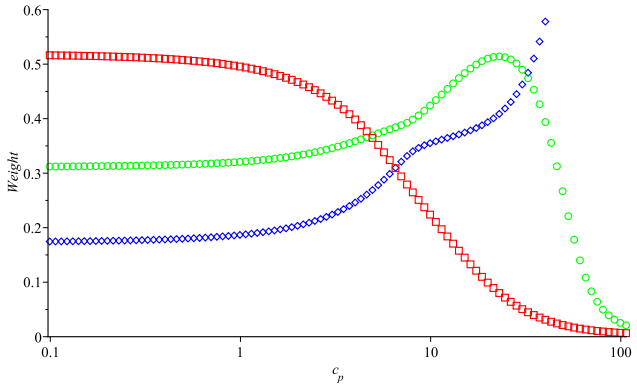

Figure 5: Weights With Increasing c_p

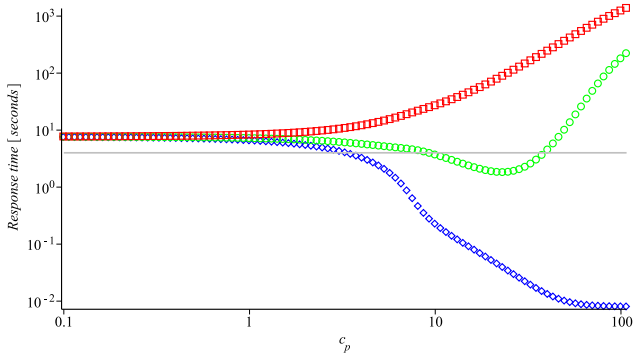

Figure 6: Weights With Increasing c_p

Figure 5 depicts the trend of the weight distribution and the according response times (cf. Figure 6) for different values of c_p. The plot shows, that for small a c_p, the response

times of all tenants are aligned. This is due to the fact, that

$$\lim_{c_v \to 0} p_j = \lim_{x \to 0} \frac{1}{1 + \exp(x)} = 1/2,$$

which means, that the penalty p_j of all tenants is approximately 1/2. Thus, no performance isolation takes place and the optimisation assigns weights, such that all response times are equal.

If the value of c_p becomes very large, i.e. $c_p \to \infty$, then the function $p_j(l_j)$ converges towards the non-continuous function.

$$p_i(l_i) := \begin{cases} 1 & \text{if } l_i < q_i \\ 1/2 & \text{if } l_i = q_i \\ 0 & \text{else.} \end{cases} \quad (6)$$

This means, that the penalty p_j of disruptive tenants, converges towards 0 for large values of c_p, whereas the penalty p_j of abiding tenants converges towards 1, if the according workload l_j is strictly below q_j. Therefore, for very high values of c_p, the resources are only divided among the abiding tenants working within their quota.

Usually a more moderate value of c_p is recommended, such that tenants are penalised more gradually as they get disruptive and where the effect of the penalty is noticeable, but not too extreme. For the given scenario a value of c_p between 1 and 10 seems to be suitable.

5. CONCLUSION

Software-as-a-Service offerings are often based on multi-tenancy where several customers share one single application instance. This leads to significantly decreasing operational costs. However, due to the tight coupling the resources are shared among all tenants which leads to undesired performance influence and hinders the SaaS Provider to scale tenants individually by means of the quality of their service. For example, in case one tenant is willing to pay additional fees for extra resources to cover an increasing demand, the provider has to ensure, these resources are only used by the tenant who requested them. Nowadays solutions often leverage request based admission control to ensure that a tenant can use the share they pay for in situations with high load. However, mapping the guaranteed response time for a given quota (e.g., max request rate) to a certain share is often a challenge in itself.

We present a fitness function which computes a numerical value based on the response time violation for each tenant. These results are used in a numerical optimizer to find a setup of weights which can be used to adjust the request admission control. A second part of the function decreases the result of the fitness function for the tenant who exceeded its quota. We proofed that this function is suitable for a numerical optimization and discussed the impact of tuning parameters. The contribution is one important step to enable performance isolation and scalability on a tenants basis.

The future work will integrate the approach in a real environment and discuss how to obtain the system performance function.

Acknowledgements

Research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 317704 (CloudScale).

6. REFERENCES

[1] TPC BENCHMARK W, 2002. Transaction Processing Performance Council.

[2] bitcurrent. Bitcurrent cloud computing survey 2011. Technical report, bitcurrent, 2011.

[3] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar. Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In *High Performance Computing and Simulation (HPCS) 2010*, 2010.

[4] C. Fehling, F. Leymann, and R. Mietzner. A framework for optimized distribution of tenants in cloud applications. In *CLOUD 2010 IEEE 3rd International Conference on Cloud Computing*, 2010.

[5] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, 2006.

[6] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley, 2004.

[7] R. Krebs, C. Momm, and S. Kounev. Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. *Elsevier Science of Computer Programming Journal (SciCo)*, 2013.

[8] R. Krebs, S. Spinner, N. Ahmed, and S. Kounev. Resource usage control in multi-tenant applications. In *CCGRID 2014 - in print*, 2014.

[9] X. Li, T. Liu, Y. Li, and Y. Chen. SPIN: Service performance isolation infrastructure in multi-tenancy environment. *Service-Oriented ComputingâĂŞICSOC 2008*, pages 649–663, 2008.

[10] H. Lin, K. Sun, S. Zhao, and Y. Han. Feedback-Control-Based Performance Regulation for Multi-Tenant Applications. In *Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems*, 2009.

[11] P. Mell and T. Grance. The NIST definition of cloud computing, 2011.

[12] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn, S. M. Vorkoetter, J. McCarron, and P. DeMarco. *Maple 10 Programming Guide*. Maplesoft, Waterloo ON, Canada, 2005.

[13] SAP AG. SAP Hana Cloud, Feb 2014. https://hana.ondemand.com.

[14] D. Smith. Hype cycle for cloud computing, 2011. Technical report, Gartner, July 2011. ID Number: G00214915.

[15] W. Wang, X. Huang, X. Qin, and W. Zhang. Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications. *2012 IEEE Fifth International Conference on Cloud Computing*, 2012.

[16] Y. Zhang, Z. Wang, B. Gao, C. Guo, W. Sun, and X. Li. An effective heuristic for on-line tenant placement problem in SaaS. *Web Services, IEEE International Conference on*, 0:425–432, 2010.