# Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments

Rouven Krebs[a,c], Christof Momm[a,d], Samuel Kounev[b,e]

[a]*SAP AG, 69190 Walldorf, Germany*
[b]*Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany*
[c]*Rouven.Krebs@sap.com*
[d]*Christof.Momm@sap.com*
[e]*Kounev@kit.edu*

**Abstract**

The cloud computing paradigm enables the provision of cost efficient IT-services by leveraging economies of scale and sharing data center resources efficiently among multiple independent applications and customers. However, the sharing of resources leads to possible interference between users and performance problems are one of the major obstacles for potential cloud customers. Consequently, it is one of the primary goals of cloud service providers to have different customers and their hosted applications isolated as much as possible in terms of the performance they observe. To make different offerings, comparable with regards to their performance isolation capabilities, a representative metric is needed to quantify the level of performance isolation in cloud environments. Such a metric should allow to measure externally by running benchmarks from the outside treating the cloud as a black box. In this article, we propose three different types of novel metrics for quantifying the performance isolation of cloud-based systems.

We consider four new approaches to achieve performance isolation in Software-as-a-Service (SaaS) offerings and evaluate them based on the proposed metrics as part of a simulation-based case study. To demonstrate the effectiveness and practical applicability of the proposed metrics for quantifying the performance isolation in various scenarios, we present a second case study evaluating performance isolation of the hypervisor Xen.

*Keywords:* Performance, Isolation, Metric, SaaS, Cloud, Multi-tenancy

## 1. INTRODUCTION

Resource sharing promises significant cost savings in cloud environments, thanks to the reduced per customer overheads and economies of scale [25] [3]. The most significant obstacle for potential cloud users, besides data isolation and security aspects, is unreliable performance [15] [3] [5]. Therefore, providing performance guarantees is a major research issue in the area of cloud computing

[12]. Providing one cloud customer constant Quality of Service (QoS) independent from the load induced by others is referred to as performance isolation.

The National Institute of Standards and Technology [23] defines three service models for cloud computing. The Infrastructure-as-a-Service (IaaS) model leverages virtualization to share hardware resources among customers. The Platform-as-a-Service (PaaS) model hosts applications of different customers within one middleware instance. Software-as-a-Service (SaaS) is the last model which provides a ready to run, hosted application. Isolating cloud customers in terms of the performance they experience is an important concern in each of these scenarios.

The allocation of hardware resources is handled by the lower levels (e.g., infrastructure level) in the stack. Therefore, we see performance isolation as a bigger challenge in the upper levels (e.g., platform and software) as they have no direct resource control. Within a SaaS environment, a group of users sharing the same view onto the application are referred to as *tenant*. This view includes the data they access, the application configuration, Service-Level-Agreements (SLAs) and Quality-of-Service (QoS) aspects. Multi-tenant Applications (MTA) share one application instance between multiple tenants and provide every tenant a dedicated share of the instance, isolated from each other. In our opinion the isolation of data and configuration aspects in MTAs is fairly easy as these issues are handled by the application domain. Usually all tenants use an application in a similar way with regards to the configuration and navigation paths. Nevertheless, the amount of users and peak times might defer and the tight coupling of tenants results in strong interference of non-functional system properties. Consequently, dealing with inference, especially considering non-functional system properties is still an open research issue in the area of SaaS (e.g., Bezemer [4], Fehling [8] and Wang [28]) and a challenging task for developers and architects of such systems.

In contrast to MTAs a hypervisor runs several virtual machine (VM) on the same hardware. A VM is a computer which is not directly accessing the hardware by leveraging virtualization. Thus, several virtual machines can run in parallel and share the resources. This technology is used to provide several customers access to SaaS offering whereby several instances of the application serve the load. Furthermore, it is the enabling technology of IaaS. In the traditional datacenters without cloud context the technology is also widely accepted and applied. We also observe considerable research interest in the field of performance isolation for IaaS clouds and virtualization technolgogies. Huber et al. [14] pointed out that different virtual machines (VM) have significant influence on each other. Gupta et al. [11] were already aware of this issue and developed an advanced scheduler resource scheduler for a specific hypervisor to solve the problem. Nevertheless, performance isolation in existing implementations has still potential to improve, especially in I/O intensive scenarios.

Performance isolation is an important aspect for various stakeholders. When a developer or architect has to develop a mechanism to ensure performance isolation between customers they need to validate the effectiveness of their approach to ensure the quality of the product. Furthermore, to improve an existing mechanism they need an isolation metric to compare different variants of the solution. When a system owner has to decide for one particular deployment in a virtual environment not only traditional questions like the separation of components on various hosts are of importance. Also the configuration of the hypervisor with regards to resource allocation mechanism have to be considered.For a concrete decision several concerns might be important. Performance, efficiency, administrative costs and security are mostly the basis for the decision. With a metric quantifying isolation, one more parameter could be used for the decision making process.

To the best of our knowledge, no metrics and techniques for quantifying performance isolation have been proposed before. In this article, we present two different methodologies and several alternative metrics along with appropriate measurement techniques for quantifying the isolation capabilities of IT systems. Although our focus is on cloud environments and cloud enabling technologies the metrics are not limited to these. The metrics presented are applicable for performance benchmarks, and preferable in situations where various customers use similar functionality with various load. In addition to this, we introduce general approaches for performance isolation in SaaS environments at the architectural level using four concrete isolation mechanisms. Finally, we apply the proposed metrics and measurement techniques for quantifying isolation in two independent case studies to on the one hand demonstrate the practical applicability of the proposed metrics. On the other hand, the metrics allow us to evaluate the effectiveness of the proposed performance isolation methods for SaaS environments and the impact of different deployment options in a virtual infrastructure that also allows us to reason for IaaS environments.

The remainder of this article is structured as follows. In Section 2, we first define performance isolation. Based on this definition, Section 3 presents the proposed isolation metrics. Section 4 discusses different approaches to ensure performance isolation within a MTA. Section 5 presents the first experiment setup for the evaluation of the isolation approaches and the metrics. Section 6 presents the second case study using virtualization. Based on the results a discussion and final assessment of the metrics and their usability in various scenarios is given in Section 7. In Section 8, we briefly introduce further ideas for enhancements of the metrics and measurement approaches. Section 9 surveys related work, while Section 10 concludes the article.

## 2. PERFORMANCE ISOLATION IN SHARED ENVIRONMENTS

This section provides a comprehensive definition of performance isolation and differentiation to related concepts in shared environments. The metrics presented in Section 3 are based on these definitions.

### *2.1. Fairness*

Performance concerns in cloud environments are a serious obstacle for consumers. To avoid distrust, it is necessary to ensure a fair behaviour of the system with respect to its different customers. Due to sharing of resources, performance-related issues are often caused by a minority of customers sending a high amount of requests. We define a system as *fair*, if the following conditions are met:

1. Customers working within their assigned quota should not suffer from customers exceeding their quotas.
2. Customers exceeding their quotas should suffer performance degradation if they have a negative impact on others. This forces them to reduce their load, which eventually restores the system responsiveness for all customers.
3. Customers with higher quotas should be provided with better performance than customers with lower quotas. Difference in performance might be expressed with regards to the output related parameters (e.g., response time) or input related parameters (e.g., request rate).

Within this article, *quota* refers to the amount of workload a customer is allowed to execute. In the following, we define performance isolation based on this notion of fairness provided by point 1.

### *2.2. Isolation*

In this article, we focus on the first fairness criterion defined above which is achieved by performance isolation. Performance isolation is defined as follows.

*Performance Isolation:* A system is performance-isolated, if for customers working within their quotas the performance is not affected when other customers exceed their quotas. A decreasing performance for the customers exceeding their quotas is fair with regard to the property 2. Additionally, it is possible to relate the definition to SLAs: A decreased performance for the customers working within their quotas is acceptable as long as it is within their SLAs. One way to achieve performance isolation is by resource isolation which enforces a strict isolation of resources allocated to different customers.

*Non-Isolation:* We speak of non-isolation, if the behaviour of the users from one customer may influence the performance observed by the users of the other customers as if all users are part of the same customer. Thus, every customer may suffer from bad performance caused by one single disruptive customer exceeding its quota.

*Firm Performance Isolation:* Motivated by the definition of real-time systems we also introduce firm performance isolation. Systems that are firmly performance isolated have one or both of the following two characteristics. Either they provide performance isolation only on a restricted set of workload scenarios. Or an observable influence of one customer onto the others exists. However, in both situations the average isolation is within the accepted range and a minor violation of isolation is acceptable within a this range.

An example for the first case is an overcommitted system which might restrict the load of a disruptive customer to the quota defined in the SLAs and thus ensures performance isolation. However, because of the overcommitment several customers might use their entire quota at one time and harm the systems overall performance. A provider might accept this risk to increase the efficiency.

*Elasticity:* Elasticiy is a concept in cloud computing to satisfy changing resource demands under variable load with the goal to maintain SLAs. To avoid the impression that isolation and elasticity have the same objectives we present the following example. Assume a system is in an overload condition because of one disruptive tenant. One approach to enforce customer SLAs could be to provision and allocate additional resources by leveraging underlying technologies. This is commonly referred to as elasticity. This could be acceptable, if the disruptive tenant pays for the increased overall capacity, e.g. in relation to its quota. If the disruptive tenant does not pay for the extra resources, the system is no longer fair. Furthermore, if new resources are provisioned they should be reserved for customers paying extra fees (fair with regard to property 3).

## 3. METRICS

Existing benchmarks and metrics in the field of shared resources and Clouds developed in the last years focus on single aspects like databases (e.g., [6]). Others discuss metrics for cloud features like elasticity (e.g., [20]) or on traditional metrics like throughput in a virtualized environment (e.g., [13]). In the following section, we introduce different metrics to quantify the isolation capabilities of a system which is not covered by the existing approaches. The actual discussion about the feasibility of the metrics is provided after the case studies in Section 7. To provide a level playing field for comparisons, it is important to explicitly consider the workload profiles used when applying the metrics. For example, a given response time for a system is meaningless without consideration of the system load during which the response time was measured. In our case further aspects like the number of customers with exceeded quota might also influence the results. In this section, we focus on the definition of adequate isolation metrics. The metrics we define may be applied to quantify the isolation of any measurable QoS-related system property in any system shared between different entities. As such, the metrics are not limited to performance isolation and Cloud although these are in the focus of this article. Of course, the actual type of workload and QoS must be selected according to the scenario under investigation. Later on in Section 5 and 6 we propose a set of specific workloads which can be used for benchmarking performance isolation. We distinguish between groups of *disruptive* and *abiding* customers. The latter work within their given quota (e.g., defined number of requests/s) the former exceed their quota. Traditional performance measurements usually have only one group of users and observe the performance of the group as a function of the workload induced by this group. Our metrics are based on the influence of the disruptive customers on the abiding customers. Thus we have to groups and observe the performance of one group as a function of the workload of the other group. This is a major differentiator of our approach to traditional performance measurements. For the definition of the metrics, we define a set of symbols in Table 1.

| Symbol | Meaning |
|---|---|
| $t$ | A customer in the system. |
| $D$ | Set of disruptive customers exceeding their quotas (e.g., contains customers inducing more than the allowed requests per second). $\|D\| > 0$ |
| $A$ | Set of abiding customers not exceeding their quotas (e.g., contains customers inducing less than the allowed requests per second).$\|A\| > 0$ |
| $w_t$ | Workload caused by customer $t$ represented as numeric value $\in \mathbb{R}_0^+$. The workload is considered to increase with higher values (e.g., request rate and job size). $w_t \in W$ |
| $W$ | The total system workload as a set of the workloads induced by all individual customers. Thus, the load of the disruptive and abiding ones. |
| $z_t(W)$ | A numeric value describing the QoS provided to customer $t$. The individual QoS a customer observes depends on the composed workload of all customer $W$. We consider QoS metrics where lower values of $z_t(W)$ correspond to better qualities (e.g., response time) and $z_t(W) \in \mathbb{R}_0^+$ |
| $I$ | The degree of isolation provided by the system. An index is added to distinguish different types of isolation metrics. The various indices are introduced later. |

Table 1: Overview of variables and symbols

## *3.1. Metrics based on QoS Impact*

QoS-oriented approaches define an isolation metric based on considering the influence of disruptive customers by measuring their impact on the QoS provided to customers working within their quotas.

These metrics depend on at least two measurements. First, the observed QoS results for every $t \in A$ at a reference workload $W_{ref}$. Second, the results for every $t \in A$ at a workload $W_{disr}$ when a subset of the customers have increased their load to challenge the system's isolation mechanisms. As previously defined $W_{ref}$ and $W_{disr}$ are composed of the workload of the same set of customers which is the union of $A$ and $D$. At $W_{disr}$ the workload of the disruptive customers is increased.

We consider the relative difference of the QoS ($\Delta z_A$) for abiding customers at the reference workload compared to the disruptive workload.

$$\Delta z_A = \frac{\sum\limits_{t \in A} [z_t(W_{disr}) - z_t(W_{ref})]}{\sum\limits_{t \in A} z_t(W_{ref})} \tag{1}$$

Additionally, we consider the relative difference of the load induced by the two workloads.

$$\Delta w = \frac{\sum\limits_{w_t \in W_{disr}} w_t - \sum\limits_{w_t \in W_{ref}} w_t}{\sum\limits_{w_t \in W_{ref}} w_t} \tag{2}$$

Based on these two differences the influence of the increased workload on the QoS of the abiding tenants is expressed as follows.

$$I_{QoS} = \frac{\Delta z_A}{\Delta w} \tag{3}$$

A low value of this metric represents a good isolation as the difference of the QoS in relation to the increased workload is low. Accordingly, a high value of the metric expresses a bad isolation of the system.

The metric provides a result for two specified workloads ($W_{ref}$ and $W_{disr}$) and thus the selection of the workloads plays an important role. On the one hand this provides a good evidence for exactly this setup and thus provides detailed information. On the other hand, only one measurement for a given workload tuple ($W_{ref}, W_{disr}$) might not be sufficient if the exact workloads of interest are unknown or vary. Thus, we enhanced the metric by considering the arithmetic mean of $I_{QoS}$ for $m$ disruptive workloads. Whereby the disruptive customers increase their workload equidistant within a lower and upper bound.

$$I_{avg} = \frac{\sum\limits_{i=1}^{m} I_{QoS_m}}{m} \tag{4}$$

This metric provides an average isolation value for the entire space of measurements and provides one representative numeric value. The curve's shape is not reflected and thus the value might lead to missleading results within some ranges.

It is conceivable that a provider is interested in the relative difference of disruptive workload $\Delta w$ at which abiding tenants receive a predefined proportion of the promised QoS $\Delta z_A$. This is conceptual similar to the already described metrics and could be used as one additional approach.

### *3.2. Workload Ratios*

The following metrics are not directly associated with the QoS impact resulting from an increased workload of disruptive customers. The idea is to compensate the increased workload of disruptive customers and try to keep the QoS for the abiding ones constant by decreasing the workload of the abiding ones. Naturally, this is only possible with the support of the abiding customers and such a behaviour does not reflect productive systems. Thus, these metrics are planned to be applied in benchmarks with artificial workloads where a load driver simulates the customers and can be enhanced to follow the described behaviour.

Assume one starts measuring the isolation behavior of a non-isolated system by continually increasing the disruptive workload $W_d$. One would expect to observe a decrease of $z_t(W)$ for all customers. In such a situation, $z_t(W)$ would remain unaffected if the workload of the abiding customers $W_a$ is decreased accordingly to compensate for the increase in the disruptive workload. Following this idea, plotting $W_a$ as a result of $W_d$ describes a pareto optimum of the systems total workload with regards to constant QoS.

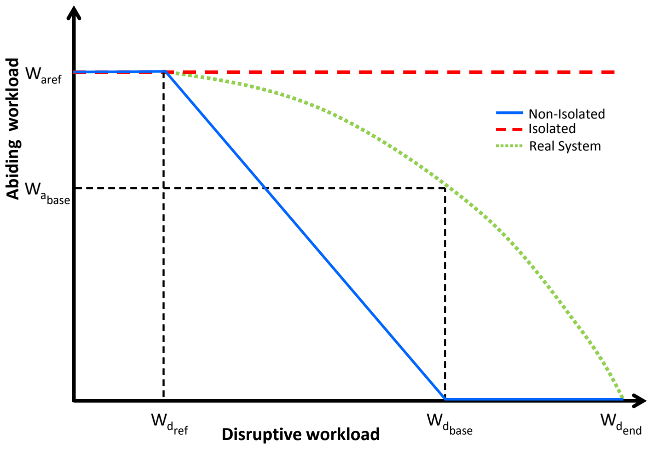


Figure 1: Fictitious isolation curve including upper and lower bounds.

In Figure 1 the x-axis shows the amount of workload $W_d$ caused by the disruptive tenants, whereas the y-axis shows the amount of the workload $W_a$ caused by the abiding tenants. The blue/solid line shows how $W_a$ has to decrease to maintain the same QoS as in the beginning. In a non-isolated system this function proportionally decrease linear. For every additional amount added to the disruptive load one has to remove the same amount at the abiding load, because in a non-isolated system the various workload groups would behave if they were one. In a perfectly isolated system the increased $W_d$ has no influence on $z_t(W)$ for all $t \in A$. Thus, $W_a$ would be constant in this case as shown with the red/dashed line in the figure. The red line and the blue line provide exact upper and lower bounds, corresponding to a perfectly isolated and a non-isolated system, respectively. Figure 1 shows some important points referenced later and defined in Table 2.

Based on this approach, we define several metrics presented in the following. As discussed before, the workload scenarios play an important role, and thus it may be necessary to consider multiple different workload scenarios and average over them as previously.

### *3.2.1. Significant Points*

The significant points marked in Figure 1 provide several ways to define an isolation metric by themselves. $I_{end}$ is a metric derived by the point at which

| Symbol | Definition |
|---|---|
| $W_d$ | The total workload induced by the disruptive customers: $W_d = \sum_{t \in D} w_t$ |
| $W_{d_{base}}$ | The level of the disruptive workload at which the abiding workload in a non-isolated system is decreased to 0 due to SLA violations. |
| $W_{d_{end}}$ | The level of the disruptive workload at which the abiding workload must decreased to 0 in the system under test |
| $W_{d_{ref}}$ | The value of the disruptive workload at the reference point in the system under test. This is the point to which the degree of isolation is quantified. It is defined as the disruptive workload, at which in a non-isolated system the abiding workload begins to decrease. |
| $W_a$ | The total workload induced by the abiding customers: $W_a = \sum_{t \in A} w_t$ |
| $W_{a_{ref}}$ | The value of the abiding workload at the reference point $W_{d_{ref}}$ in the system under test. $W_{a_{ref}} = W_{d_{base}} - W_{d_{ref}}$ |
| $W_{a_{base}}$ | The value of the abiding workload corresponding to $W_{d_{base}}$ in the system under test. |

Table 2: Overview and definition of relevant points.

the workloads of abiding customers have to be decreased to 0 to compensate for the disruptive workload. The metric sets $W_{d_{end}}$ and $W_{a_{ref}}$ in relation. Due to the discussed relationship of the workloads in a non-isolated system and the definition of the various points based on the behaviour of such a system the condition $W_{a_{ref}} = W_{d_{base}} - W_{d_{ref}}$ holds. We leverage this relation to simplify our formulas. With Figure 1 in mind, $I_{end}$ is defined as follows:

$$I_{end} = \frac{W_{d_{end}} - W_{d_{base}}}{W_{a_{ref}}} \tag{5}$$

Another approach uses $W_{a_{base}}$ as a reference. Setting this value and $W_{a_{ref}}$ in relation results in an isolation metric having a value between $[0, 1]$. The formula for metric $I_{base}$ is below:

$$I_{base} = \frac{W_{a_{base}}}{W_{a_{ref}}} \tag{6}$$

Both metrics have some drawbacks resulting from the fact that they do not take into account the curve progression. This means, that in a system which behaves linearly until a short distance from $W_{d_{base}}$ and then suddenly drops to $W_a = 0$, both metrics would have the same value as in the case of a completely non-isolated system which is obviously unfair in this case. Moreover, a well isolated system might require a very high disruptive workload before $W_a$ drops to 0 making it hard to measure the metric in an experimental environment. $I_{base}$ has some further disadvantages given that it is only representative for the behavior of the system within the range of $W_{d_{ref}}$ and $W_{d_{base}}$. Given that the metric does not reflect what happens after $W_{d_{base}}$, it may lead to misleading results for well isolated systems whose respective $W_{d_{end}}$ points might differ significantly.

For systems that exhibit a linear degradation of abiding workload, we could also define isolation metrics based on the angle between the observed abiding workloads line segment and the line segment which represents a non-isolated system. However, linear behaviour typically cannot be assumed.

*3.2.2. Integral Metrics*

We define two further isolation metrics addressing the discussed disadvantages of the above metrics. They are based on the area under the curve derived for the measured system $A_{measured}$ set in relation to the area under the curve corresponding to a non-isolated system $A_{nonIsolated}$. The area covered by the curve for a non-isolated system is calculated as $W_{a_{ref}}^2/2$.

The first metric $I_{intBase}$ represents the isolation as the ratio of $A_{measured}$ and $A_{nonIsolated}$ within the interval $[W_{d_{ref}}, W_{d_{base}}]$. We define $f_m : W_d \to W_a$ as a function which returns the residual workload for the abiding customers based on the workload of the disruptive customers. We then define the metric $I_{intBase}$ as follows:

$$I_{intBase} = \frac{\left( \int\limits_{W_{d_{ref}}}^{W_{d_{base}}} f_m(W_d)\, dW_d \right) - W_{a_{ref}}^2/2}{W_{a_{ref}}^2/2} \tag{7}$$

$I_{intBase}$ has a value of 0 in cases the system is not isolated and a value of 1 if the system is perfectly isolated within the interval $[W_{d_{ref}}, W_{d_{base}}]$. The metrics major advantage is, that the value provided allows to set the system directly into relation to an isolated and non-isolated system. This metric again has the drawback that it only captures the system behavior within $[W_{d_{ref}}, W_{d_{base}}]$.

In a well isolated system it might not be feasible to measure the system behavior only up to $W_d base$. Thus, the following metric $I_{intFree}$ allows to use any predefined artificial upper bound $p_{end}$ which represents the highest value of $W_d$ that was measured in the system under test. We define the metric as follows:

$$I_{intFree} = \frac{\left( \int\limits_{W_{d_{ref}}}^{p_{end}} f_m(W_d)\, dW_d \right) - W_{a_{ref}}^2/2}{W_{a_{ref}} \cdot (p_{end} - W_{d_{ref}}) - W_{a_{ref}}^2/2} \tag{8}$$

This metric quantifies the degree of isolation provided by the system for a specified maximum level of injected distructive workload $p_{end}$. A value of 1 represents a perfect isolation and a value of 0 a non-isolated system.

## 4. PERFORMANCE ISOLATION IN MULTI-TENANT APPLICATIONS

Regarding performance, related work is mostly concerned about resource efficiency and optimal placement of tenants onto a limited set of nodes with regards to their SLAs (e.g., Fehling et al. [8] and Zhang [32]).

This section discusses different approaches to enforce isolation within a multi-tenant application. We start with an overview of a multi-tenant architecture presented by Koziolek [18] and discuss related performance isolation extensions. Following this, we present some detailed approaches to enforce performance isolation.

### *4.1. Multi-tenant Software Architectures*

Koziolek [18] [17] analyzed several existing multi-tenant solutions. Based on this, he developed a generic architecture and a corresponding style describing the existing multi-tenant applications. Figure 2 presents Koziolek's architecture. The numbers were added by us and present points we identified as positions the system performance can be influenced. The architecture relies on the common three tier web application model enhanced with a Meta-Data Manager and a Meta-Data Database responsible for tenant-specific customization regarding the functional/non-functional behavior and/or appearance of the application.

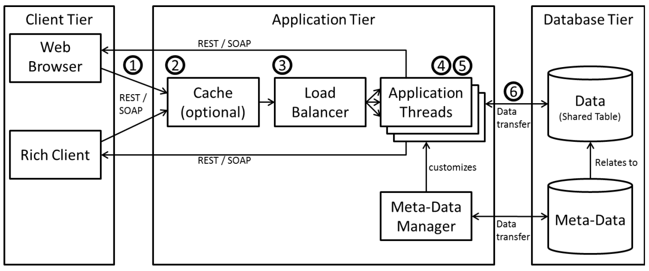


Figure 2: Potential points for a performance adaptation in multi-tenant architectures based on [18] [17].

*Admission Control(1):* Controlling the incoming requests based on awareness of the tenant they originate from can enable a MTA to differentiate the provided QoS level and to isolate tenants. One solution could be to discard requests sent by a disruptive tenant in order to maintain the service level for the abiding tenants.

*Cache Restrictions (2):* In some MTA, cached objects might be shared between tenants. Some applications (like multimedia platforms) could use high amounts of tenant specific data. Restricting the size of the cache available to tenants is one way to isolate them from using space reserved for others.

*Load Management (3):* The ability of a MTA to serve different tenants with one instance does not prevent us from having several application instances available. Thus, a load balancer could enforce some isolation by forwarding requests of disruptive tenants to different application instances than those of abiding tenants.

*Thread Priorities (4) and Thread Pools (5):* The main computing power is consumed by threads handling requests. Thus, it is possible to implement isolation approaches by leveraging thread management functionality. For example, using a separate thread pool for every tenant limits the number of threads one tenant could allocate at a time. Another idea is to dynamically control priorities of threads depending on the tenant.

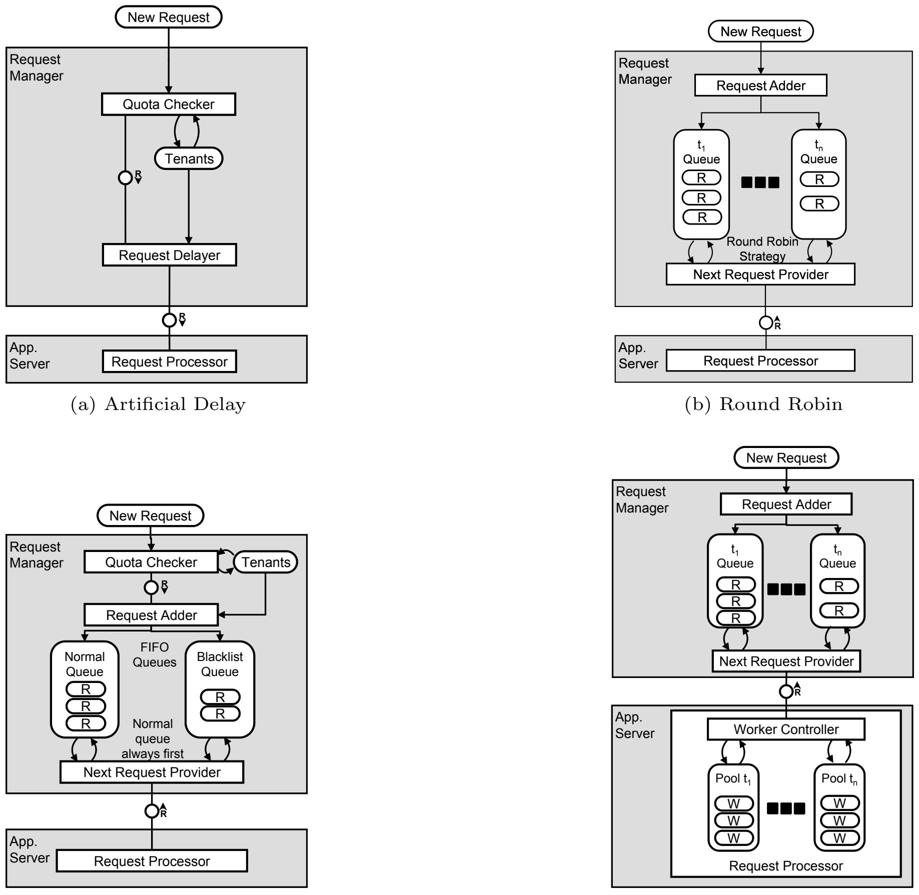
*Database admission (6):* Controlling the incoming database requests based on the tenant they originate from can enable a MTA to differentiate the provided QoS level and to isolate tenants. Controlling tenant specific database cache sizes at the application level is another approach to isolate tenants.

### *4.2. Performance Isolation Solutions*

In this section, we propose four approaches, developed in the context of our work, for enforcing performance isolation focusing on response time as primary QoS metric. The application area is a multi-tenant, interactive web application where we assume a rather similar behaviour of different tenants and request types as no batch jobs are expected and fast response times required. These concepts leverage on admission control and thread pool management mechanisms which corresponds to the position 1 and 5 in Figure 2. In the following, you will find figures, explaining the basic architectural structure of the different approaches. Every approach implements two top level components: A Request Manager handling the incoming request and an Application Server providing the Request Processor. In the default case, the Application Server's Request Processor has one thread pool with restricted size processing the requests.

*Artificial Delay:* This approach (Figure 3a) artificially delays incoming requests depending on the request rate of the corresponding tenant. In closed workload scenarios this results in artificially increased response times for tenants exceeding their quotas and generates backpressure. Thus, the overall workload induced by a tenants is controlled. A new request arrives the Request Manager's Quota Checker which evaluates the tenants currently used quotas, and stores the results together with the allowed quotas in the tenants meta data. After that, the quota checker triggers the request delayer, which possibly delays the processing of a request before it is forwarded to the Request Processor for processing. The duration of the artificial delay could be constant. The current demand of the system and the difference between the allowed and actual usage of the system could be used to calculate a dynamic delay. Within the application server, the request might be FIFO-queued again, because of the restricted size of the thread pool.

*Round Robin:* Round robin (Figure 3b) introduces separate queues for different tenants. There is no more need for a queue at the application server in this scenario, as requests are directly buffered at the Request Manager. When a new request approaches the system, it is queued in the corresponding tenant's queue. If the Request Processor has free threads, it triggers the Next Request Provider to deliver a new request. The Next Request Provider then uses round



(a) Artificial Delay (b) Round Robin

(c) Blacklist (d) Thread Pool

Figure 3: Methods for performance isolation in multi-tenant applications.

robin to retrieve the next request. An empty queue for one tenant is skipped and does not block the processing of the others.

*Blacklist:* The blacklist method (Figure 3c) triggers the quota checker for every request. It checks if the quota for this particular tenant is exceeded. The quotas available to tenants and the quotas actually used by them are maintained in the tenants meta data. If a tenant exceeds its quota, it is blacklisted. Requests from blacklisted tenants are enqueued in a separate list. When the Request Processer requests for the next request, the Next Request Provider takes the next request from the white queue on a FIFO basis. Usually, requests from the blacklist queue are only handled if the normal queue is empty. This leads to a problem, when a tenant is removed from the blacklist but he has requests still pending in the blacklist queue. If there are always requests in the white queue, blacklisted requests will never be handled. We realized a mechanism that slowly processes requests from the blacklist (e.g., every 30th request). The method takes the first request, at which the tenant is actually not blacklisted anymore.

*Separate Thread Pools:* The separate thread pool method (Figure 3d) provides a separate thread pool for each tenant. The limited size of these pools isolates the tenants from each other. The conceptual model includes a separate FIFO queue for each tenant. Every time, one of the tenant specific thread pools has an idle thread, the Worker Controller requests a new request from the Next Request Provider. The Next Request Provider selects a pending request according from the tenants thread pool.

## 5. SIMULATION BASED EVALUATION

This section presents the results of a simulation-based evaluation. This allows us to evaluate different concepts for isolation and the metrics efficiently without disturbing influences. Furthermore, it is an example how a developer or architect might use the metrics to decide for one implementation.

### *5.1. Simulation*

We employed the ssj[1] discrete event simulation framework including the provided stochastic features [21] as a simulation allows us to evaluate different concepts and the metrics efficiently without disturbing influences. The major artifacts that were used to simulate our shared system are the *RequestManager*, *RequestProcessor*, *Tenant* and *Scheduler*. The *RequestManager* is responsible to realize the different approaches for performance isolation discussed previously and checks if the *RequestProcessor* has free resources to forward the next request. If no resources are available, the request is queued until the *RequestProcessor* signals that resources became free again. The *RequestProcessor* is responsible to simulate the request processing behavior according to the predefined scheduling strategy. The *Scheduler* used within our evaluation simulates a resource which is partially shared and assumes that the capacity of requests it can handle is limited. Thus, the calculation of the residual service time for each request is based on the number of requests in the *RequestPocessor* and a user defined factor for the proportion of shared resources as well as a reference service time for the request in an unused system. The value becomes actualized every time a request arrives or leaves the *RequestProcessor*. Once the request is processed it is sent back to the corresponding *Tenant*. The *Tenant* instance than simulates the think time and initializes the request for the next iteration.

### *5.2. Evaluation Scenarios*

In this section, we present the workload profiles, the performance related QoS of interest and the configuration we have chosen for our evaluation.

[1]http://www.iro.umontreal.ca/s̃imardr/ssj/indexe.html

### *5.2.1. QoS-Metric and Considered Workload*

The QoS metric we focus on is the response time. The time is measured from the moment a request leaves a tenant to the point in time a tenant receives the response. Thus, $z_t(W)$ returns the average response time for $t$. As a measure for the workload caused by the tenants, the number of users associated with each tenant is used.

The workload profile we used is described by the users' behavior, the type of requests sent, the amount of tenants in each group $D$ and $A$, and the number of users associated with each tenant.

In a MTA the workload induced by the tenants is rather homogeneous (except the amount). In our simulation, all users send requests of the same type with a mean think time of 1000ms and a standard deviation of 100ms in a closed workload scenario. We expect that the system runs with a high utilization for economic reasons. Another reason for running under high utilization is our goal of evaluating performance isolation aspects. In a system with low utilization, the increased workload of one tenant would have low impact. Therefore, we designed our system to serve total 80 users. The mean service time for a request in the system without contention is 1000ms with a standard deviation of 150ms.

We consider one normal scenario and one with overcommitment. In the first scenario, the quota is set to 8 users and in the overcommitted one to 24. In both situations, we expect only one disruptive tenant ($t_0$). The number of users in the first scenario is 8 and in the overcommitted one $t_0 = 24, t_1..t_3 = 8, t_4 = 4, t_5..t_8 = 1, t_9 = 24$.

Thus, the total workload was set to a value at which the system is already at its limit of 80 users and the disruptive tenant allocates its full quota. We consider this to be the best reference point. First, an increased workload of one tenant in a non-isolated system would immediately cause SLA violations. Second, with the next increase of workload by the disruptive tenant the isolation mechanisms should intervene.

For the QoS-oriented metrics, we also have to define the disruptive workloads. For $t_0$ we have chosen 24, 40 and 251 users in the normal mode. In the overcommitted scenario the number of users are set to 40, 56 and 251. For the averaged isolation metric $I_{avg}$ we did measurements beginning with 8 and stopping with 248 users in the normal scenario and 24 to 264 users in the overcommited scenario. The values were increased by a stepwidth of 40 users. In the following, we indicate the number of users by adding indexes to the various symbols in order to distinguish the results.

### *5.2.2. Configuration*

In the chosen configuration with a standard, non-tenant aware FIFO queue as *RequestManager*, the maximum throughput achieved is 18 requests/second at a response time of 2110ms (Figure 4). This results in 38 requests handled in parallel. Thus, the size of the thread pool is restricted to 38 threads for an optimal throughput. Without a restricted thread pool, most of the presented performance isolation approaches would fail, as the *RequestManager* would always forward the requests to the processor. When 80 users are simulated, a standard FIFO queue results in an average response time of 3500ms and 62 requests in the system, whereby 24 are queued.

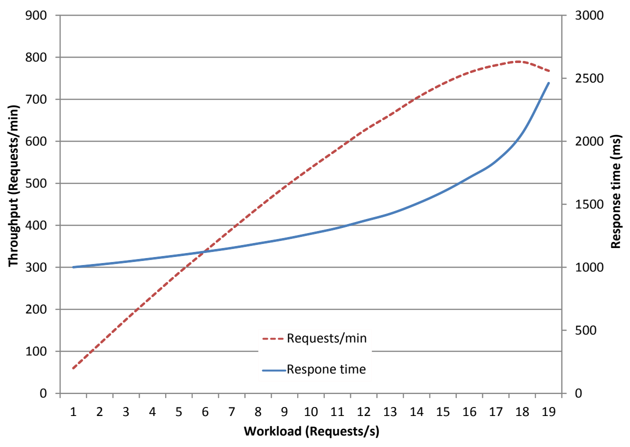


Figure 4: Measurement of Throughput and Response Time.

## *5.3. Evaluation Results*

In this section, we present the results from the simulation described above and briefly comment on the observations made in the various considered scenarios. The overall assessment follows in a separate section.

Exemplary we calculate $I_{qos_{24}}$ for the normal non isolated case and $I_{intFree_{251}}$ for the delay method in the overcommitted scenario. In the non-isolated case the simulation returns a response time of $3446ms$ at the reference workload of 8 users for $t_0$ and $4334ms$ at the disruptive workload with 24 users for $t_0$. Due to the absence of isolation the average response time for the abiding customers is the same as for the disruptive customers. This results in $\Delta z_{A_{24}} = \frac{4334-3446}{3446} = 0.258$. The relative increase of workload is $\Delta w = \frac{96-80}{80} = 0.2$. Consequently the isolation metric is calculated as $I_{qos_{24}} = \frac{0.258}{0.2} = 1.29$.

In the delayed scenario with $I_{intFree_{251}}$ the point $p_{end}$ is at 251, $W_{d_{ref}}$ is 24. The integral desribing the area below the curve of remaining abiding users $\int f(W_d)dW_d$ within the limits $[24, 251]$ is directly deduced from the measurements (Figure 5) and has a value of 4687. $W_{a_{ref}}$ was set to 56 in the workload definition. Thus, $W_{a_{ref}}^2/2 = 1568$ and consequently $I_{intFree_{251}} = \frac{4687-1568}{56\cdot(251-24)-1568} = 0.28$

Concerning the overview of all results we begin with the QoS related metrics presented in Table 3. The value for the isolation in the non-isolated situation is almost the same in every case as the impact on the performance is linear, because it stems from the extended length of the queue.

Table 4 present the integral related metrics for the different approaches and workloads. The $n/a$ entries stem from a very high value of $W_{d_{end}}$ which was

| | Normal | | | | Overcommitted | | | |
|---|---|---|---|---|---|---|---|---|
| Approach | $I_{QoS_{24}}$ | $I_{QoS_{40}}$ | $I_{QoS_{251}}$ | $I_{avg248}$ | $I_{QoS_{40}}$ | $I_{QoS_{56}}$ | $I_{QoS_{251}}$ | $I_{avg264}$ |
| Non-Isolated | 1.29 | 1.29 | 1.29 | 1.29 | 1.29 | 1.29 | 1.29 | 1.29 |
| Round Robin | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.02 | 0.06 | 0.03 |
| Thread Pools | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.00 | -0.01 | 0.00 |
| Delay | 0.32 | 0.59 | 1.22 | 0.9 | -0.49 | 0.19 | 1.22 | 0.67 |
| Black List | 0.09 | 0.10 | 0.01 | 0.03 | -0.73 | -0.26 | 0.02 | -0.03 |

Table 3: Results of QoS based metrics.

not in the range of our evaluation. The rest of the section discusses different behaviors of the isolation methods and their impact onto the metrics aligned with selected conspicuous measurements.

| | Normal | | | | Overcommitted | | | |
|---|---|---|---|---|---|---|---|---|
| Approach | $I_{end}$ | $I_{base}$ | $I_{intBase}$ | $I_{intFree_{251}}$ | $I_{end}$ | $I_{base}$ | $I_{intBase}$ | $I_{intFree_{251}}$ |
| Non-Isolated | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Round Robin | n/a | 1 | 1 | 1 | n/a | 1 | | 0.99 |
| Thread Pools | n/a | 1 | 1 | 1 | n/a | 1 | 1 | 1 |
| Delay | 1.11 | 0.58 | 0.68 | 0.23 | 1.5 | 0.75 | 0.86 | 0.28 |
| Black List | n/a | 0.94 | 0.96 | 0.97 | n/a | 0.96 | 0.94 | 0.96 |

Table 4: Results of workload ratio based metrics.

*Round Robin:* This method provides a good isolation in every scenario. In the chosen scenario, the waiting queue for the disruptive tenant was never empty at the reference workload. Therefore, $t_0$ was not able to disrupt the other tenants by increasing its workload. In cases where at the reference workload the queue of the disruptive tenant runs empty, the increased load is expected to influence the others tenants.

*Separate Thread Pools:* In the normal mode, the size of the thread pools was set to 4 which results in around 38 allowed threads in the system. To keep the response time for the tenants with 24 users lower than 3500ms we had to increase the thread pool to 17 threads. This resulted in an overloaded situation, but thanks to the reduced queueing time we still achieved a response time of 3.5s at the reference workload. Overall, the thread pool approach showed a very good isolation.

*Artificial Delay:* The threshold for the artificial delay was based on the number of users logged in for one tenant. The negative values of $I_{QoS_{40}}$ and $I_{QoS_{56}}$ stem from the constant penalty added to every request arising from the disruptive tenants. Therefore, a part of the disruptive tenants' resources become available for the other tenants and consequently the QoS for the abiding tenants improves. This results in negative isolation values. The isolation works only within a limited range because of the constant character of the delay. This point can be seen in Figure 5 when the abiding workload begins to decrease.

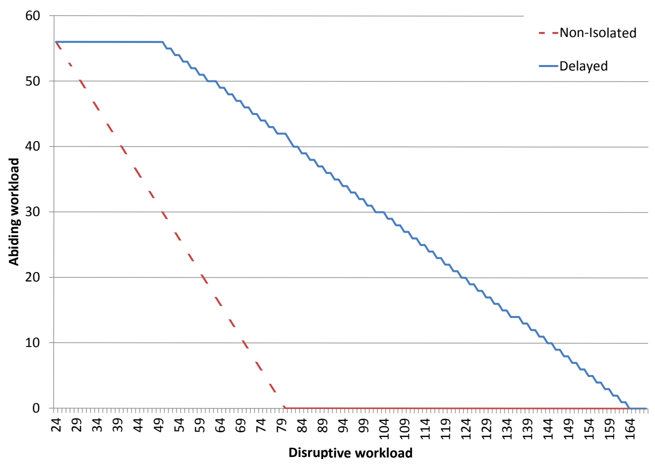


Figure 5: Reduction of abiding workload while artificial delay is activated in the overcommitted scenario.

*Blacklist:* The blacklist exhibits a similar behavior as the artificial delay in the beginning. Therefore, a negative isolation was measured. The relevant metric for blacklisting the tenant is its throughput. The raw data shows, that sometimes a white tenant is also blacklisted for a short while. This occurs in situation when the actual disruptive tenant becomes blacklisted and its part of the resources become available for the other tenants. In that situation the response time and consequently the request rate of the abiding tenants improves and exceeds the quota. However, the effect is negligible with regards to the average response times.

## *5.4. Effectiveness of the SaaS Isolation Methods*

The evaluation of the isolation methods presented in 4.2 is primarily based on the results from our simulation study. Additionally we highlight some other aspects that struck our attention.

Round robin provides a very good isolation. However, it is not sufficient for overcommitted systems as it cannot fully leverage the unused resources from some tenants. It is possible to increase a tenant's throughput by skipping empty queues. However, as long as requests are pending in every queue, tenants with more users have higher response times. In our case the tenants with 24 users had around 4660ms response time, even in cases in that the total amount of users was within the limits in which the system could provide 3500ms for every tenant. This is an issue in overcommitted systems. Furthermore, it is not possible to provide different QoS to different tenants using a simple round robin. Thread pools achieve good isolation in the simulations and QoS differentiation is achievable by using different thread sizes. Besides that, the processing speed and throughput of tenants is better if some tenants do not allocate all of their threads. This also increases the throughput. In the chosen overcommitted scenario, the response time is widely constant over all tenants at the reference workload. A disadvantage is that the total number of potential threads is above the optimal working point of the server. Thus, there is a danger of congesting the server.

The delay approach seems to be ineffective, because of its weak isolation. A dynamically assigned delay could increase effectiveness with the drawback of increased complexity of the method. The introduction of different thresholds enables QoS differentiation. The blacklist approach provides a good isolation over a wide range of disruptive workloads. Furthermore, it could achieve different QoS, especially for throughput, by using different thresholds. Additionally, unused resources are equally used by all tenants. For the abiding tenants, in the overcommitted scenario, this results in the same response time (around 3500ms) for each. In our setup, the response time at $W_{disr_{56}}$ was around 9360ms for $t_0$ where the thread pool is used. In the black list approach, it was around 16040ms. In both cases, the mean response times for the other tenants were around 3500ms. This stems from the abiding tenants that were blacklisted from time to time resulting in unstable performance.

## 6. VIRTUALIZATION BASED CASE STUDY

This section presents the results of a virtualization based case study. By this, we evaluate the applicability of the metrics in real environments and give some insights on the isolation capabilities of the widely used hypervisor Xen. Furthermore, it is an example how the metrics can be used by system owners to decide for a deployment scenario.

Beside multi-tenancy, the sharing of hardware resources by serving several operating systems on the same host is a widely adopted technology and the foundation for IaaS clouds. Xen [2] is a widely used hypervisor for Linux environments that enables resource sharing on hardware level. Thus, we decided to stress Xen with regards to performance isolation by leveraging our previously described approach. More precisely, we quantify the degree of isolation for various Xen configurations and deployments based on a black box approach using our previously defined metrics. Therefore, we deploy several instances of the TPC Benchmark W (TPC-W) [1] onto different VMs hosted by one Xen hypervisor and measure how they influence each other. Precisely because, the case study itself has not a concrete Cloud deployment in mind it shows the wide range of scenarios supported by the metrics and the results still allow to reason for the isolation capabilities of IaaS clouds running on Xen.

In the following, we describe some details of Xen, the chosen TPC Benchmark W and the system landscape, followed by the scenario specific configuration and finally the results with a short discussion.

### *6.1. Xen*

A Hypervisor is a software to run several virtual machines (VM) as guests on one host. Xen is one of the most known hypervisors for Linux environments. The operating systems installed within these VMs are decoupled from the other systems, have no permission for administrative tasks on the hardware or the hypervisors configuration. In order to configure the system or the hypervisor and to execute administrative tasks the first VM started in Xen (domain-0 or dom0) has special privileges. Furthermore, dom0 provides a driver abstraction for the different guest systems. The drivers in Xen are divided in two parts. The driver really accessing the hardware is installed in dom0, the guest systems (domU) drivers communicate with the dom0 to access the hardware. Consequently dom0 might become a bottleneck for various activities. Especially I/O intensive tasks are known to produce high overhead in dom0 and thus the independent guest domains are likely to influence each other on these tasks. Such a behavior was already observed in [14] and [11]. By default the various VMs have access to all existing resources. To increase performance and isolation it is possible to exclusively pin a core to a domain. It is worth to mention that dom0 usually does not host any services for the actual end user due to its administrative role.

### *6.2. TPC-W*

The TPC-W [1] is a benchmark for business oriented transactional web servers. The workload is based on a controlled internet commerce environment and simulates a bookshop. In our setup, the bookshop consists of a Java

Servlet based application and a SQL database. The benchmark simulates multiple on-line browser sessions by calling dynamically generated pages. These servlets access the database and consistent web objects. Usually the performance metric reported by TPC-W is the number of web interactions processed per second. However, in our case we consider the average response time of the requests. The benchmark simulates three profiles that differ by the browse to buy request ratio: primarily shopping, browsing and web-based ordering. In our case study we used the browsing workload. The load can be varied by the amount of emulated browsers (EB) running against a system. One EB simulates one user calling various web transactions in a closed workload. Based on the benchmark's heavy I/O demands we expect to observe an influence of the various domains onto each other.

## *6.3. System Landscape*

The physical landscape comprises two servers with 2 physical quad core CPUs (2133Mhz and 2 threads for each core) and contains 16 GiB main memory. On both servers Xen 4.1 is installed and Suse Linux Enerprise (SLES) 11 SP2 is running on dom0 and on the guest systems. The servers are connected with a 1Gbit ethernet link. One server hosts the load driver for the TPC-W benchmark in dom0. The various domains of the second server are described at the scenario specific configuration 6.4.

The database schema is refreshed before every measurement and filled with 100000 items and 300000 customers.

At our first measurements we observed that increasing the load by more than a maximum specific to the systems configuration results in timeout exceptions or socket/file handle issues. Thus, a further increase of the load is no longer representative, because the induced demand is no longer equivalent to the load induced before this maximum was reached. Consequently, it does not represent the corresponding demand for the abiding domains anymore. Therefore, we had to tweak our system in several ways. We configured the application servers http timeouts to be infinite, the operating systems socket specific timeouts to be around 6 minutes and the maximum number of open TCP connections was increased to the operating systems maximum value. Besides this, one has to avoid domain internal (software) bottlenecks, because this hinders the system to increase the load for the shared hardware resources under investigation. Therefore, several measurements to find the optimum thread pool size and connection pool limits were done before the actual isolation measurements of each scenario.

## *6.4. Evaluation Scenarios*

In total we investigated three different scenarios in this case study. In the *pinned* scenario the server hosts 4 guest systems (dom1, dom2, dom3, dom4) and dom0. Every domU has a fixed memory allocation of 3096MB and hosts a MySQL 5.0 database and a SAP specific customized Tomcat webserver. The various domains were exclusively pinned to the existing cores. Thus, no competition for the same CPU resources was possible. Based on this runtime environment four separate instances of the TPC-W bookshop application were deployed.

In the *unpinned* scenario all domU and the dom0 were not pinned to a specific CPU and free to use any existing hardware resource. Xen's credit scheduler was chosen to allocate the domains to the various resources.

In addition to this we investigated an *unpinned two tier* scenario, which also doesn't have a fixed CPU pinning and likewise uses the Xen credit scheduler. However, the database and the application server in this case are deployed onto separate domains. Every domU with an application server has a fixed memory allocation of 2024MB and the database domain allocates 1024MB. This memory setup was chosen, because of the small database size.

Table 5 shows the values we used to define the reference and disruptive workloads for the various scenarios. The number of EBs at the maximum accumulated throughput of all domains is presented in the second column, the corresponding accumulated throughput, per domain specific throughput and average response times are listed next. The last column shows the disruptive domains amount of EBs at which we observed a high proportion of failed requests. In the unpinned two tier scenario we observed different values for the QoS based and workload ratio based metrics.

| Scenario | EBs per domU | Total throughput | Throughput per domU | Avg. response time | Max. load disruptive |
|---|---|---|---|---|---|
| Pinned | 3000 | 1195 r/s | 299 | 1104ms | 15000 |
| Unpinned | 1500 | 721 r/s | 180 | 842ms | 13500 |
| Unpinned two tier | 1300 | 617 r/s | 154 | 833ms | 8000 (QoS) 11050 (ratios) |

Table 5: Results for the scenario setup and configuration.

The highest difference in throughput for one domain compared to the mean was around 4.5% and the highest difference of the response times around 6.5% in the pinned scenario. In the unpinned case we observed 2.2% (one tier) and 2.7% (two tier) difference for the throughput. The difference of the response times was at 8.2%(one tier) and 9.4% (two tier).

As a consequence of these observations $p_{end}$ is set to 15000 for the pinned scenario and to 13500 for the unpinned. In the unpinned two tier scenario we had to choose 11050 for $p_{end}$ and had to stop our evaluation for the $I_{QoS}$ metrics at 8000 user. It is worth to mention that in both unpinned scenarios $p_{end}$ is very close to nine times the load of the maximum throughput for one domain.

## *6.5. Evaluation Results*

In this section we provide an overview of the measurement results and the observed isolation metrics. Figure 6 combines the results for both unpinned scenarios based on normalized values for the abiding and disruptive load. Table 6 presents the QoS based metrics based on the same values of $\Delta w$. Thus, the results provides a comparable view onto both deployments. The detailed discussion of the results shown in the table and figure follows in the discussion of the concrete scenarios.

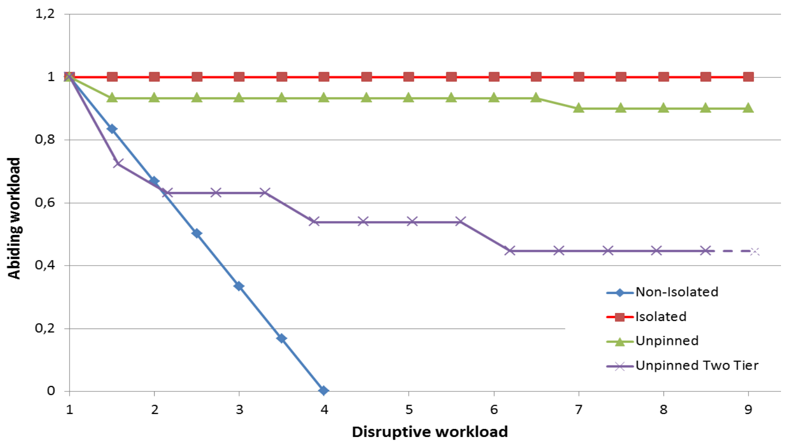


Figure 6: Normalized reduction of abiding workload in the unpinned and unpinned two tier scenario.

Table 6 contains the values of $I_{QoS}$ for all three scenarios. The first column of Table 6 identifies the scenario, the second the amount of users for the disruptive domain, the third column the average response time of all abiding domains followed by the results for $\Delta w$, $\Delta z$, $I_{QoS}$ and $I_{avg}$. For the pinned scenario we did only one measurement due to the very good isolation. To ensure a level playing field of comparison we selected measurements where $\Delta w$ is the same in the relevant scenarios. The $I_{avg}$ values were calculated based on an interpolation of the measurements of $\Delta w$ at 0.33, 0.60, 1.05 in the unpinned two tier scenario and additionally 1.47 in the unpinned scenario. For the pinned scenario we assume a linear behaviour of the isolation between $\Delta w = 0$ and $\Delta w = 4$.

### *6.5.1. Pinned*

Overall, this scenario presented a nearly perfect isolation throughout the whole range. The $I_{QoS}$ presented in Table 6 at a disruptive load of 15000 users was below 0.05 and the $I_{avg}$ resulted in 0.04 The workload ratio based metric decreased for the abiding workload only once at 12000 disruptive users. The related metrics $I_{intFree15000}$ and $I_{intBase}$ resulted in a value short below 1.

| Scenario | Disruptive load | Response time | $\Delta w$ | $\Delta z$ | $I_{QoS}$ | $I_{avg}$ |
|---|---|---|---|---|---|---|
| Pinned | 15000 | 1317 | 4.00 | 0.19 | 0.05 | 0.03 |
| Unpinned | 3200 | 927ms | 0.33 | 0.10 | 0.30 | 0.18 |
| | 4800 | 942ms | 0.60 | 0.12 | 0.20 | |
| | 7500 | 914ms | 1.05 | 0.09 | 0.08 | |
| | 10000 | 1173ms | 1.47 | 0.39 | 0.27 | |
| Unpinned two tier | 3000 | 1011ms | 0.33 | 0.21 | 0.64 | 4.20 |
| | 4400 | 3784ms | 0.60 | 3.54 | 5.90 | |
| | 6750 | 4354ms | 1.05 | 4.22 | 4.02 | |

Table 6: Results of $I_{QoS}$ in the various scenarios.

### *6.5.2. Unpinned*

For the metrics based on the QoS impact we determined the isolation at various disruptive workloads shown in table 6. We observed two significant characteristics. The first one is the increasing response time when the disruptive load is set to 3200 users. The second is the increasing response time at 10000 users. Accordingly, the isolation becomes better between 3200 users and 10000 users. This is, because of the widely constant response times by increasing load which changes the ratio of $\Delta z/\Delta w$. In average the isolation ($I_{avg}$) is 0.18.

Figure 6 presents the total abiding workload $W_a$ based on the disruptive users. Similar to the $I_{QoS}$ based results two significant points can be observed at the same position. In both cases, $W_a$ decreased because of an increasing response time at the abiding tenants. At a disruptive load of 13500 users (corresponds to 9 in the figure) the disruptive domain failed to successfully handle incoming requests. Therefore the results are no longer valid for higher disruptive loads. The overall isolation values are $I_{intFree13500} = 0.89$ and $I_{intBase} = 0.86$.

### *6.5.3. Unpinned Two Tier*

Table 6 shows the various disruptive loads used to evaluate $I_{QoS}$. We configured the disruptive loads in a way, they result in the same $\Delta w$ as in the unpinned single-tier scenarios. Due to the increasing number of timeouts and exceptions at the disruptive domain we had to stop at 6750 users. For this workload range, we observed a continuous increasing response time. Nevertheless, from 4400 users to 6750 users the isolation became better, because $\Delta w$ increased more than $\Delta z$. Over the whole range of the measurements the average isolation $I_{avg}$ is 4.20.

Figure 6 presents the total abiding workload $W_a$ based on the disruptive users for the workload ratio based metrics. Analogous to the response times in table 6 we can see a continuous decreasing amount of abiding workload in Figure 6. At 1.5 in the figure we can see the observed isolation curve crossing the characteristic of a non-isolated system. This is, due to the selected step width for reducing the number of users at the disruptive domain. At a disruptive load of 11050 users (corresponds to 8.5 in the figure) the disruptive domain failed to successfully handle incoming requests. These results are no longer valid for higher disruptive loads and are therefore illustrated using a dashed line. The overall isolation values are $I_{intFree1105} = 0.42$ and $I_{intBase} = 0.36$.

### *6.6. Effectiveness of the Deployment Options*

Overall, the *pinned* scenario showed the best results and the *unpinned two tier* the worst. The selected size of the database was small enough to be mostly cached. The memory was not overcommitted in our setup and the network I/O did not reach a critical point at which the CPUs for dom0 became a bottleneck in the one tier scenarios. Therefore, the isolation was nearly perfect with pinned CPUs. In the *unpinned* scenario the resources of the domU became shared with those for dom0, therefore the slightly increasing I/O overhead for dom0 was competing for resources and had some minor effect. The credit scheduler was not able to completely compensate this. By splitting the dom0 into an application server and database server we visibly increased the network I/O. In this setup we observed a significant impact of the disruptive domain onto the others, whereby the handling of the I/O in dom0 led to a bottleneck or requested additional processing resources from the guest domains.

When an administrator has to decide for one of the mentioned deployments, various considerations might be of importance. In a pinned setup the overall performance and isolation is the best. However unused resources of one domain cannot be used by other domains, thus this setup might lack in efficiency. The *unpinned* scenario overcomes this drawback but at the expenses of performance and isolation. From a separation of concerns point of view it might be beneficial to separate database and application. On the other hand, a distributed deployment is less performant as table 5 shows and the isolation is the worst. The case study showed how the isolation metrics provide the opportunity to quantify one more dimension in the framework of several trade-off questions a system provider has to answer. An additional result is, that an administrator can increase isolation by hard resource allocation and deployments which reduce I/O. Furthermore this allows to reason for IaaS Cloud environments, that applications with high I/O demands are less isolated than others.

## 7. FINAL ASSESSMENT OF THE METRICS

In the following we evaluate the metrics, regarding their expression and feasibility.

For the evaluation of the metrics we concentrate on the following aspects. First, how feasible is the metric for the target group of a system owner/provider or a developer/researcher. Second, the expressiveness of the metric in terms of the type of evidence it provides. Third, the number of measurements required to obtain a valid value. Fourth, situations in which the metric is not meaningful.

## *7.1. QoS Impact*

These metrics show the influence of disruptive workloads on the QoS of abiding tenants. This helps system owners to manage their systems, because it indicates the influence of disruptive workload onto the QoS they provide, which is important for capacity planning. QoS-based metrics can prove that a system is perfectly isolated, however they fail in ranking a systems isolation capabilities into the range between isolated and non-isolated. Thus it is hard to estimate the potential of the method. In the simulation shown, we were able to measure the system's behavior in a non-isolated case. In reality this is rarely possible, as a system owner or user might not be able to change, or event set off, the system's isolation method (e.g., in our case study). A single $I_{QoS}$ metric can be derived with only two measurements to obtain evidence for one point of increased workload. However, to obtain some more detailed information concerning the systems isolation more measurements are required. Therefore, $I_{avg}$ describes the average isolation value within the upper and lower bound of interest. Nevertheless, the metric is not suitable to describe a systems behavior for different disruptive workloads onto the abiding tenants because it cannot be set into a relation for a concrete scenario. Thus we see the advantage of this metric rather in comparing different systems.

## *7.2. Significant Points*

The metric $I_{end}$ might not be feasible to quantify isolation in well isolated systems. Furthermore, it is not possible to directly deduce relevant system behaviors like response times. If the metric is given, it could help to compare two systems regarding the maximum disruptive load they can handle. To determine $I_{end}$, more measurements as for QoS-based metrics are required.

$I_{base}$ orders a system within the range of isolated and non-isolated systems for one specific point in the diagram. Nevertheless, it does not provide information about the behavior of the system before that point. It is limited to comparing the isolation behavior of the systems at one selected load level and it is inadequate to derive direct QoS-related values. The usefulness of this metric appears to be of limited value in contrast to the integral methods. One advantage is the evidence at a well-defined and reliable point with only two measurements.

## *7.3. Integral Metrics*

$I_{intBase}$ and $I_{intFree}$ are widely comparable metrics. $I_{intBase}$ has the advantage to be measured at a predefined point. For $I_{intFree}$, the endpoint of the interval must be considered as well to have an expressive metric. Both metrics provide good evidence of the isolation within the considered interval, ordered between the magnitudes of isolated and non-isolated systems. They lack in providing information concerning the degree of SLA violation. For example, the SLA violation could be very low and acceptable or critically high in each iteration when we reduce $W_a$. However, in both cases, the results of the metrics are similar. This limits the value of $I_{intBase}$ and $I_{intFree}$ for system owners/providers. However, for comparison of systems and analyzing their behavior, the metrics are very useful and can be exploited by developers or researchers. Finally, on the negative side, a disadvantage of these metrics is that their measurement may be a time consuming task. In our Xen based case study we had experiment series of around 15 hours.

### *7.4. Concluding Discussion of the Metrics*

The various metrics show their advantages in different fields of application and express various semantics. The $I_{QoS}$ and $I_{avg}$ metrics represent the reduced QoS based on disruptive load. It can not provide a ranking within the range of isolated and non-isolated systems. However, for a system operator this might be helpful to estimate the impact of disruptive load onto the system. The $I_{end}$ metric shows how many times a system is better than a non-isolated one. This information is helpful to compare different systems if one has to decide for one. The Integral based metrics rank a system within the range of isolated/non-isolated. This knowledge is beneficial for the developer of a system to estimate the potential for improvements.

## 8. FURTHER ENHANCEMENTS

Within this chapter we discuss three practice oriented improvements for the measurement methods described previously to broaden their scope.

### *8.1. Various Workloads*

Workload can be divided in two parts. The work defines the tasks and the sequence they are processed by the system. The load defines the amount. In the selected evaluation scenarios we varied the load of the customers/tenants. In a multi-tenant environment, where every tenant uses the same application it is a valid assumption that the type of work generated by different tenants is rather equivalent. In virtualized environments, the workloads might be more heterogeneous. It is entirely conceivable that a provider or customer is interested to measure the influence of various workload types onto each other. One example is the impact of an I/O intensive application onto a CPU intensive one.

In such cases the function for a non-isolated system might not follow the rules described in 3.2. Even the definitions for the abiding and disruptive load could be of a completely different nature.

To solve this problem we propose to setup a non-isolated environment (e.g., deploying two different applications within one operating system). With such a setup the curve measured by the workload ratio approaches represents the non-isolated curve. Furthermore, it enables us to set the two different workloads into relation to each other and it provides a way to calculate the overall workload needed for the QoS impact oriented metrics.

### *8.2. Speed Up of Measurement Progress*

The reduction of the time needed to collect meaningful data in an adequate amount is essential to make a performance measurement method succeed. Within our evaluation we observed a runtime of around 90 minutes for the integral metrics at the simulation. Whereby the hardware used, was a standard desktop PC with 2 cores and 4GB memory. In the use cases with the Xen hypervisor we had already observed, that the runtime for measuring every single manipulation of the amount of user is not realistic and used intervals of various step widths. However, even in this case one measurement for the integral metrics used around 15 hours and the accuracy was not optimal, because of the interval length. In some use cases an accurate measurement might need to much time.

The adaptive breakdown measurement strategy presented in [19], [31] provides a mechanism to reduce the amount of experiments required. An experiment is defined as one run of the benchmark with one defined setup. The adaptive breakdown method consists of four basic steps.

The first selects a set of configurations for experiments with the goal to increase the accuracy of an interpolation the most. The second starts the benchmark with the defined experiments and collects the results. The third creates an interpolation based on the supporting points. In the fourth step, validation experiments are selected and executed to find the area, where the interpolation error is the maximum. After this last step the process is iteratively repeated.

In [19] we already observed good results of this method in scenarios where only one parameter (disruptive workload in our case) is varied. As interpolation method a linear interpolation can be used. Once the function is derived it is possible to calculate the integral.

### *8.3. Measurement Framework*

Manual performance measurements of computer systems is an error prone and time intensive task. The Software Performance Cockpit (SoPeCo) [29] [30] goal is to automate the performance measurements process. Therefore the framework provides several adapters to apply the same measurement techniques to various platforms. As aforementioned the metrics defined in this article are flexible and might be adapted for any system where resources are shared. However, implementing the algorithms to adjust the workloads and the calculation of the metrics is a time intensive task. Therefore, the SoPeCo can be leveraged to decouple the measurement logic and metrics from the actual domain and technical constraints. Thus, the adaption of the presented metrics becomes easily realizable.

## 9. RELATED WORK

We divided the related work into two parts. The first is about related work in the area of the defined isolation metrics. The second covers related work in the field of performance isolation in MTA.

### *9.1. Metrics*

The lack of performance guarantees is one of the major obstacles in cloud computing [3] [5]. As a result different benchmarks and metrics were developed in the last years. Usually these publications focus on single aspects of cloud services like databases (e.g., [6]). Others discuss metrics for cloud features like elasticity (e.g., [20]). However, the most relevant related work we found, comes from the field of virtualization, which is the main enabling technology for IaaS.

One industrial example is VMmark [13], a benchmark developed by VMWare. They define a tile as a set of VMs serving different applications (e.g., mail server and SPECweb2005). Several tiles are deployed on a virtualized hardware. The benchmark score is based on a normalized overall throughput of the applications, a hosting platform could achieve. The total throughput increases with the number of tiles deployed as long as the system is not saturated. VMWare publishes the number of tiles in addition to the throughput. However, VMmark focuses on overall performance of a hosting platform and fails to quantify the mutual influence, of the different workloads.

Georges et al. [9] developed two metrics to express the efficiency of a virtualized environment. One similar to VMmark. The other, Average Normalized Reduced Throughput (ANRT), reflects the loss of throughput on a per VM basis, when additional VMs are deployed. Nevertheless, they do not set the amount of changed workload in relation to ANRT and use static amount of workload for the VMs. Thus, these metrics are not feasibly to be used for quantifying performance isolation.

Koh et al. [16] collected data within an experimental environment to closely characterize the performance inference of workloads in different VMs. In addition, a prediction mechanism was implemented to predict the inference of these workloads. Huber et al. [14] created a feature tree capturing the mutual influences of different VMs with different resource requirements. This was done in an automated way.

Nevertheless, Huber and Koh did not extract a single value describing the systems isolation behavior which might be used within a benchmark. Furthermore, their approaches focus on hardware related resources, only available in white box scenarios. Consequently the approaches are hard to be used in SaaS or PaaS scenarios.

Guo et al. [10] defined performance isolation based on their understanding of a fair system behavior. From their point of view a system should prevent high performance for one tenant at the cost of another. In cases the SLAs of the tenants differ, providing different performance is still seen as fair. However, our definition of fairness was explicitly divided into three different aspects. Additionally, we see performance isolation as only one part of a fair behavior. Ensuring different SLAs on the system is important and is integrated as one aspect in our definition of fairness although it does not directly relate to performance isolation.

### *9.2. Performance Isolation in MTAs*

The popularity of multi-tenancy arose with the increasing interest in SaaS applications. Several publications focus on general aspects of MTA, their requirements and potential implementations [4] [10] [24].

Regarding performance, related work is mostly concerned about resource efficiency. Fehling et al. [8] analyzed the challenges arising from multi-tenant scenarios and provided a method to place tenants onto locations with different QoS. Zhang [32] developed a method to place on boarding tenants on a restricted set of nodes without SLAs violations. A good placement helps to decrease the interference. However, it cannot completely ensure isolation.

Schroeter et al. [26] present a tenant aware component model which allows automated reconfiguration. This might be leveraged to ensure isolation by placing a disruptive tenant onto single nodes or by adding resources (elasticity). Nevertheless, performance isolation is not in the focus of this article and based on our definition, elasticity does not automatically ensure isolation.

An approach to achieve performance isolation within MTAs was proposed by Lin et al. [22]. They provide different QoS on a tenant's base. Additionally, one test case evaluated the system regarding tenant specific workload changes and their interference. Two proportional-integral controllers were used to achieve this. The first one ensures the average overall response time by regulating the request rate; the second one leveraged different thread priorities to control the response times for different tenants. However, this approach was built mainly with the goal to differentiate QoS. Although it provides minimal admission rate settings for each tenant, the rates used are far below the system's limit. No evaluation was done for scenarios where the tenant quotas were sized to work at a saturated system. Furthermore, compared to the above approach our proposed isolation methods reduce the complexity of the implementation.

Wei et al. [27] developed a resource isolation mechanism based on a detailed resource demand estimation for the tenants. The isolation was achieved by blocking requests from tenants who had negative influence onto the others. The presented approach differs from ours in several points. First, our approaches are rather static, without dynamic adaptions to the system state. Second, the focus of our work is directly coupled with SLAs. Controlling resources does not guarantee that SLAs are fulfilled and the mapping between SLAs and resources is still an open research question (e.g., Emeakaroha [7]). Furthermore, our approaches have no additional overhead for the calculation of the resource demands.

## 10. CONCLUSION

Cloud environments are becoming widely adopted due to their cost efficient way of providing resources. However, performance isolation is still a widely open issue, especially for SaaS offerings. To make different solutions or evolutions of the same solution comparable one needs a metric to quantify the isolation capabilities of systems.

This article presents two different approaches and three basic metrics, for quantifying performance isolation, decoupled from a concrete scenario and evaluated in the context of multi-tenant SaaS applications and a virtualized infrastructure. The first one is based on the impact of an increased workload, from one customer, on the QoS of other customers. This metric has strengths to express the impact of workload on the QoS which is relevant for capacity planning. The second group of metrics does reduce the workload of the customers working within their quota ($W_a$), if the workload of the disruptive customers increases. This maintains constant QoS for the residual workload of $W_a$. One subgroup of metrics relies on resulting significant points (e.g., when $W_a$ becomes 0), another one on the area under the curve of $W_a$. The results show strengths of these metrics in ordering a system between the magnitudes of isolated and non-isolated which makes systems easily comparable.

Furthermore, the article presents different approaches to achieve performance isolation within a multi-tenant application. In addition, we realized four approaches within a simulated environment. The subsequent discussion showed that either round robin for scheduling requests of different tenants or blacklisting disruptive tenants is a suitable approach.

The case studies showed how a system provider or developer could use our metrics to quantify, and thus to improve, various performance isolation approaches. In the second case study we showed how these metrics can be applied to help administrators to find a suitable deployment for a system in a virtual environment. Furthermore, the virtualization based case study allows to reason for the isolation capabilities of IaaS Clouds.

Our future research goals are targeted at developing advanced isolation approaches which address all fairness aspects discussed in the beginning of the article and consider multiple server instances. Furthermore, we will investigate the implications of different request types on the isolation methods and metrics in more detail.

## 11. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement № 258862.

## References

[1] *http://www.tpc.org/, TPC-W*, webpage, 2012.

[2] *http://xen.org/, xen.org*, webpage, 2012.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, *Above the clouds: A berkeley view of cloud computing*, Tech. Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[4] Cor-Paul Bezemer and Andy Zaidman, *Multi-tenant SaaS applications: maintenance dream or nightmare?*, Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE) (New York, NY, USA), IWPSE-EVOL '10, ACM, 2010, pp. 88–92.

[5] bitcurrent, *Bitcurrent cloud computing survey 2011*, Tech. report, bitcurrent, 2011.

[6] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears, *Benchmarking cloud serving systems with YCSB*, Proceedings of the 1st ACM symposium on Cloud computing (New York, NY, USA), SoCC '10, ACM, 2010, pp. 143–154.

[7] Vincent C. Emeakaroha, Ivona Br, Michael Maurer, and Schahram Dustdar, *Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and*, SLA parameters in Cloud environments. The 2010 High Performance Computing and Simulation Conference (HPCS 2010) June 28July 2, 2010, 2010, pp. 48–54.

[8] C. Fehling, F. Leymann, and R. Mietzner, *A framework for optimized distribution of tenants in cloud applications*, Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, 2010, pp. 252 –259.

[9] Andy Georges and Lieven Eeckhout, *Performance metrics for consolidated servers*, HPCVirt 2010, 2010.

[10] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao, *A framework for native multi-tenancy application development and management*, E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on, 2007, pp. 551 –558.

[11] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat, *Enforcing performance isolation across virtual machines in xen*, Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware (New York, NY, USA), Middleware '06, Springer-Verlag New York, Inc., 2006, pp. 342–362.

[12] Michael Hauck, Matthias Huber, Markus Klems, Samuel Kounev, Jrn Mller-Quade, Alexander Pretschner, Ralf Reussner, and Stefan Tai, *Challenges and opportunities of cloud computing*, Karlsruhe Reports in Informatics 19, Karlsruhe Institute of Technology - Faculty of Informatics, 2010.

[13] Bruce Herndon, Paula Smith, Lisa Roderick, Eric Zamost, Jennifer Anderson, Vikram Makhija, Bruce Herndon, Paula Smith, Eric Zamost, and Jennifer Anderson, *Vmmark: A scalable benchmark for virtualized systems*, Tech. report, VMware, 2006.

[14] Nikolaus Huber, Marcel von Quast, Michael Hauck, and Samuel Kounev, *Evaluation and modeling virtualization performance overhead for cloud environments*, Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011), Noordwijkerhout, The Netherlands, May 7-9 2011, pp. 563 – 573.

[15] IBM, *Dispelling the vapor around cloud computing*, Whitepaper, IBM, IBM CorporationNew Orchard RoadArmonk, NY 10504 U.S.A., January 2010, Produced in the United States of AmericaJanuary 2010All Rights Reserved.

[16] Younggyun Koh, R. Knauerhase, P. Brett, M. Bowman, Zhihua Wen, and C. Pu, *An analysis of performance interference effects in virtual environments*, Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on, april 2007, pp. 200 –209.

[17] Heiko Koziolek, *Towards an architectural style for multi-tenant software applications*, Proc. Software Engineering (SE'10), LNI, vol. 159, GI, February 2010, pp. 81–92.

[18] Heiko Koziolek, *The sposad architectural style for multi-tenant software applications*, Proc. 9th Working IEEE/IFIP Conf. on Software Architecture (WICSA'11), IEEE, July 2011, pp. 320–327.

[19] Rouven Krebs, *Combination of measurement and model based approaches for performance prediction in service oriented systems*, Master's thesis, University of Applied Sciences Karlsruhe, Germany, Nov 2010.

[20] Michael Kupperberg, Nikolas Herbst, Joakim Kistowski, and Ralf Reussner, *Defining and quantifying elasticity of resources in cloud computing and scalable platforms*, Tech. report, Karlsruhe Institute of Technology, 2011.

[21] Pierre L'Ecuyer and Eric Buist, *Simulation in java with SSJ*, Proceedings of the 37th conference on Winter simulation, WSC '05, Winter Simulation Conference, 2005, pp. 611–620.

[22] Hailue Lin, Kai Sun, Shuan Zhao, and Yanbo Han, *Feedback-control-based performance regulation for multi-tenant applications*, Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems (Washington, DC, USA), ICPADS '09, IEEE Computer Society, 2009, pp. 134–141.

[23] Peter Mell and Timothy Grance, *The NIST definition of cloud computing*, digital, 2011.

[24] R. Mietzner, T. Unger, R. Titze, and F. Leymann, *Combining different multi-tenancy patterns in service-oriented applications*, Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International, sept. 2009, pp. 131 –140.

[25] Christof Momm and Rouven Krebs, *A qualitative discussion of different approaches for implementing multi-tenant SaaS offerings*, Proceedings of Software Engineering 2011 (SE2011), Workshop(ESoSyM-2011), 2011.

[26] Julia Schroeter, Sebastian Cech, Sebastian Goetz, Claas Wilke, and Uwe Assmann, *Towards modeling a variable architecture for multi-tenant SaaS-applications*, Proceedings of Sixth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '12), 2012.

[27] Wei Wang, Xiang Huang, Xiulei Qin, Wenbo Zhang, Jun Wei, and Hua Zhong, *Application-level cpu consumption estimation: Towards performance isolation of multi-tenancy web applications*, Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, june 2012, pp. 439 –446.

[28] Zhi Hu Wang, Chang Jie Guo, Bo Gao, Wei Sun, Zhen Zhang, and Wen Hao An, *A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing*, e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on, oct. 2008, pp. 94 –101.

[29] Dennis Westermann and Jens Happe, *Performance Cockpit: Systematic Measurements and Analyses*, ICPE'11: Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering (New York, NY, USA), ACM, 2011.

[30] Dennis Westermann, Jens Happe, Michael Hauck, and Christian Heupel, *The performance cockpit approach: A framework for systematic performance evaluations*, Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010), IEEE Computer Society, 2010, pp. 31–38.

[31] Dennis Westermann, Rouven Krebs, and Jens Happe, *Efficient Experiment Selection in Automated Software Performance Evaluations*, Lecture Notes in Computer Science (LNCS) volume 6977 (EPEW2011), Springer, 2011.

[32] Yi Zhang, Zhihu Wang, Bo Gao, Changjie Guo, Wei Sun, and Xiaoping Li, *An effective heuristic for on-line tenant placement problem in SaaS*, Web Services, IEEE International Conference on **0** (2010), 425–432.