Chapter 16
Performance Isolation

Rouven Krebs and Samuel Kounev

Cloud computing enables resource sharing at different levels of a data center infrastructure. Hardware and software resources in a data center can be shared based on server virtualization, application containerization, or multi-tenant software architectures.

Multi-tenancy is an approach to share one application instance among multiple customers by providing each of them with a dedicated view. Tenants expect to be isolated in terms of the application performance they observe; therefore, a provider's inability to offer performance guarantees can be a major obstacle for potential cloud customers. A *tenant* is a group of users sharing the same view onto an application. This view includes the data they access, the application configuration, the user management, application-specific functionality, and related non-functional properties. Usually, the tenants are members of different legal entities. This comes with restrictions (e.g., concerning data security and privacy). In this chapter, multi-tenancy is understood as an approach to share an application instance between multiple tenants by providing every tenant with a dedicated share of the instance isolated from other shares with regard to performance, appearance, configuration, user management, and data privacy (Krebs et al., 2012). Some publications use a broader definition of the term multi-tenancy; however, in this chapter, we focus on the case of shared application instances as described above.

Hypervisors and virtual machines provide another way of sharing resources between customers. In contrast to multi-tenant applications, a hypervisor runs multiple virtual machines (VMs) on the same hardware. By leveraging virtualization technology, the VMs can run in parallel and share the underlying physical resources. This technology is used to provide multiple customers access to Software-as-a-Service (SaaS) offerings whereby several instances of an application are used to serve user requests. Furthermore, virtualization is an enabling technology for Infrastructure-as-a-Service (IaaS) where customers rent VMs from cloud providers.

Despite the use of shared resources, users expect to have the feeling of control over their own and separate environment, with their own Service-Level Agreements (SLAs) and regulations as known from private data centers, both in virtualized and multi-tenant application scenarios. In addition, they expect to be *isolated* from other customers with regard to functional and non-functional aspects. However, due to the sharing of resources, performance-related issues may appear when a customer

S. Kounev et al., *Systems Benchmarking*, https://doi.org/10.1007/978-3-030-41705-5_16

sends a high number of requests generating load on the system. This is because the load generated by one customer competes for resources also used by others. Especially in the cloud context, where resources are shared intensively among customers, it is not easy to maintain reliable performance. This is a serious obstacle for cloud customers, especially for users of multi-tenant applications.

This chapter presents metrics to quantify the degree of performance isolation a system provides. The metrics are based on Krebs et al. (2014), Krebs (2015), and Herbst, Bauer, et al. (2018), and they have been endorsed by the SPEC Research Group (Herbst, Krebs, et al., 2016). In an ideal case, one should be able to measure performance isolation externally, that is, by running benchmarks from the outside and treating the system as a black box. This enables their use for a broad set of applications given that no internal knowledge of the system is required. The metrics and the thought process to create them serve as a practical example illustrating the metric attributes and principles introduced in Chapter 3.

The metrics presented in this chapter are applicable for use in performance benchmarks that measure the performance without requiring internal knowledge. They are preferable in situations where different request sources use the functions of a shared system with a similar call probability and demand per request but with a different load intensity. These characteristics are typical for multi-tenant applications but can also occur in other shared resource systems. This chapter introduces the metrics and provides a case study showing how they can be used in a real-life environment.

16.1 Definition of Performance Isolation

To avoid distrust in a multi-tenant application provider, it is necessary to ensure fair behavior of the system with respect to its different tenants. It is assumed that each tenant is assigned a *quota* that specifies the maximum load the tenant is allowed to place on the system, for example, the maximum number of service requests that can be sent per second. In this chapter, the following definition of *fairness* is used:

Definition 16.1 (Fairness) A system is considered to be *fair* if all of the following conditions are met:

1. Tenants working within their assigned quotas must not suffer performance degradation due to other tenants exceeding their quotas.
2. Tenants exceeding their quotas may suffer performance degradation; tenants exceeding their quotas more should suffer higher performance degradation than tenants exceeding their quotas less.
3. Tenants exceeding their quotas may suffer performance degradation only if other tenants that comply with their quotas would otherwise be affected.

The term *quota* refers to the amount of workload a tenant is allowed to execute (e.g., number of user requests or transactions per second).

Tenants working within their quotas will be referred to as *abiding tenants*, whereas tenants that exceed their quotas will be referred to as *disruptive tenants*. The term *guarantee* refers to the negotiated performance level as part of SLAs with the provider. The main focus of this chapter is on the first fairness criterion, which is achieved by performance isolation.

Definition 16.2 (Performance Isolation) Performance isolation is the ability of a system to ensure that tenants working within their assigned quotas (i.e., abiding tenants) will not suffer performance degradation due to other tenants exceeding their quotas (i.e., disruptive tenants).

A system is usually expected to be somewhere in between being completely performance isolated and non-isolated. A system where the influence of a tenant on other tenants is lower is considered to provide a better performance isolation compared to a system where the influence is higher.

SLAs are of major importance for shared services. Therefore, it may be useful to reflect this in the previous definitions. This would imply that the performance of tenants working within their quotas is allowed to be reduced as long as the guaranteed level of performance is maintained. The latter is essential in order to allow overcommitment of resources. Note that, in a non-isolated system, the guaranteed performance for abiding tenants eventually will be violated if the disruptive tenants continue to increase their workload. In contrast, an isolated system will maintain the guaranteed performance independent of the disruptive tenants' workload.

16.2 Performance Isolation Metrics

The performance isolation metrics we present in this chapter are not necessarily coupled to performance and they do not express the system's capability to accomplish useful work. They rather express the influence a tenant has on the ability of another tenant to accomplish useful work.

Existing benchmarks and metrics in the field of shared resources and cloud computing focus on specific aspects like database performance (Cooper et al., 2010). Some works discuss metrics for cloud features like elasticity (Herbst, Kounev, et al., 2013; Islam et al., 2012; Kupperberg et al., 2011), as discussed in Chapter 15, or performance variability (Iosup, Ostermann, et al., 2011; Schad et al., 2010). Performance variability characterizes the changes in performance over time while the workload is assumed to be constant. However, these changes are not set in relation to the workload induced by others and thus a new approach is required.

In the following, the goals and requirements for the new isolation metrics are discussed. After that, the definitions of the metrics are presented. A case study measuring performance isolation in virtualized environments serves as an example showing the metrics in action. Based on the practical experiences from the case study, we then perform a final assessment of the usability of the metrics before concluding this chapter.

16.2.1 Metrics Goals and Requirements

To improve an existing performance isolation mechanism, application developers need isolation metrics in order to compare different variants of an isolation approach. For stakeholders involved in operations, the impact an increasing workload has on other tenants can be of interest in order to define SLAs or to manage the system's capacity.

As per our definition, a system is performance isolated if each tenant working within his quota is not negatively affected in terms of performance when other tenants increase their workloads beyond their quotas. A decreased performance for the tenants exceeding their quotas is fair with regard to the second fairness property (see Section 16.1). Moreover, as mentioned earlier, it is possible to link the definition of performance isolation to the assumed performance guarantees using SLAs. As a result, a decreased performance for tenants working within their quotas would be acceptable as long as it is within their SLA-defined guarantees. These aspects have to be reflected by performance isolation metrics.

The metrics should be designed to support answering the following questions:

Q1 How much can a tenant's workload influence the performance of other tenants?
Q2 How much potential exists for improving a system's performance isolation?
Q3 Which performance isolation technique is better?

Besides these metric-specific requirements, several general quality attributes and criteria for good metrics were introduced in Chapter 3 (Section 3.4.2): *ease of measurement*, *repeatability*, *reliability*, *linearity*, *consistency*, and *independence*.

For the measurement of performance isolation, one has to distinguish between groups of *disruptive* and *abiding* tenants as defined in Section 16.1. The presented metrics are based on the influence of the disruptive tenants on the abiding tenants. Thus, the influence on one group as a function of the workload of the other group must be evaluated. This is a major difference to traditional performance benchmarking. For the definition of the metrics, a set of symbols is defined in Table 16.1.

The metrics presented in the rest of this chapter can be applied to quantify isolation with respect to any measurable QoS-related property of a system that is shared between different entities. As such, the metrics are not limited to performance isolation in multi-tenant applications, although the latter are used as an example in this chapter.

Assume a non-isolated system and the situation illustrated in Figure 16.1 where disruptive tenants increase their workload over time. Assuming that the system is not isolated, the response time for the abiding tenants and their users would increase in the same way as if these users would belong to the disruptive tenants.

Table 16.1: Overview of variables and symbols for performance isolation metrics

Symbol	Meaning
D	Set of disruptive tenants exceeding their quotas (i.e., tenants inducing more than the allowed maximum requests per second); in the context of measuring performance isolation, we assume that $\|D\| > 0$
A	Set of abiding tenants not exceeding their quotas (i.e., tenants inducing less than the allowed maximum requests per second); in the context of measuring performance isolation, we assume that $\|A\| > 0$
t	A tenant in the system; we assume that $t \in D$ or $t \in A$
w_t	The workload caused by tenant t represented as a numeric value in $\mathbb{R}_0^+$; the value is considered to increase with higher loads on the system (e.g., request rate or job size); $w_t \in W$
W	The total system workload as a set of the workloads induced by all individual tenants
$z_t(W)$	A numeric value describing the Quality-of-Service (QoS) (e.g., request response time) provided to tenant t; the individual QoS a tenant observes depends on the aggregate workload W of all tenants; QoS metrics with lower values of $z_t(W)$ correspond to better QoS (e.g., faster response time); $z_t : W \to \mathbb{R}_0^+$
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; a numeric suffix to the index is used in some places to express the load level under which the isolation is measured)

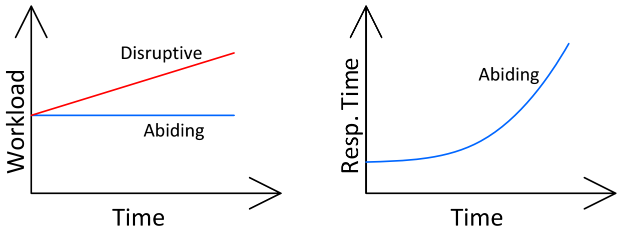


Fig. 16.1: Influence of disruptive tenants on the response time of abiding tenants in a non-isolated system

16.2.2 QoS-Impact-Based Isolation Metrics

QoS-impact-based isolation metrics depend on at least two measurements: First, the observed QoS for every abiding tenant $t \in A$ at an application-wide reference workload W_{ref}; second, the QoS for every abiding tenant $t \in A$ at a modified workload W_{disr} where a subset of the tenants have increased their load to challenge the system's isolation mechanisms. W_{ref} and W_{disr} are composed of the aggregate workload of the same set of tenants, that is, the union of A and D. In W_{disr}, the workload of the disruptive tenants is increased.

The relative difference in the QoS for abiding tenants at the reference workload compared to the disruptive tenant workload can be computed as

$$\Delta z_A = \frac{\sum\limits_{t \in A} \left[z_t(W_{disr}) - z_t(W_{ref}) \right]}{\sum\limits_{t \in A} z_t(W_{ref})}. \tag{16.1}$$

The relative difference of the load induced by the two workloads is given by

$$\Delta w = \frac{\sum\limits_{w_t \in W_{disr}} w_t - \sum\limits_{w_t \in W_{ref}} w_t}{\sum\limits_{w_t \in W_{ref}} w_t}. \tag{16.2}$$

Based on these two quantities, the influence of the increased workload on the QoS of the abiding tenants is expressed as follows:

$$I_{QoS} := \frac{\Delta z_A}{\Delta w}. \tag{16.3}$$

A low value of this metric represents a good isolation, as the impact on the QoS of abiding tenants in relation to the increased workload is low. If the value is 0, the isolation is perfect. Accordingly, a high value of the metric indicates a bad isolation of the system. In principle, the upper bound of the metric is unlimited. A negative value may occur if a mechanism reduces the performance of the disruptive tenants more than expected, thus providing the abiding tenants an even better performance.

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. However, only one measurement for a given workload tuple (W_{ref}, W_{disr}) is not sufficient if the exact workloads of interest are unknown or variable. To address this, one can consider the arithmetic mean of I_{QoS} for m different disruptive tenant workloads as follows:

$$I_{avg} := \frac{\sum\limits_{i=1}^{m} I_{QoS_m}}{m}. \tag{16.4}$$

This metric provides an average isolation value for the entire considered space of workloads and provides one representative numeric value. The disruptive tenant workload is increased in equidistant steps within a lower and upper bound. However, the curve's shape is not reflected in the average value and it may thus lead to misleading results for some ranges of disruptive tenant workload.

It is conceivable that a provider might be interested in the relative difference of disruptive tenant workload Δw at which abiding tenants receive a predefined proportion Δz_A of the promised QoS. This is conceptually similar to the already described metrics and could be used to extend them with further metrics.

16.2.3 Workload-Ratio-Based Isolation Metrics

The metrics we introduce in the following are not directly associated with the QoS impact resulting from an increased workload of disruptive tenants. Instead, the idea is to compensate for the increased workload of disruptive tenants by decreasing the workload of the abiding ones such that the QoS for abiding tenants can remain unaffected. Figure 16.2 illustrates this. For simplicity, we assume that in a non-isolated system, resources are equally shared among the tenants; therefore, the response time would maintain a constant value if abiding tenants decrease their workload by the same amount as the amount by which disruptive tenants increase theirs. The better the performance isolation, the less abiding tenants would have to reduce their workload. Naturally, this is only possible with the support of the abiding tenants and such a behavior would not be expected in productive systems. Thus, these metrics are planned to be applied in benchmarks with artificial workloads where a load driver simulates the tenants and can be programmed to follow the described behavior.

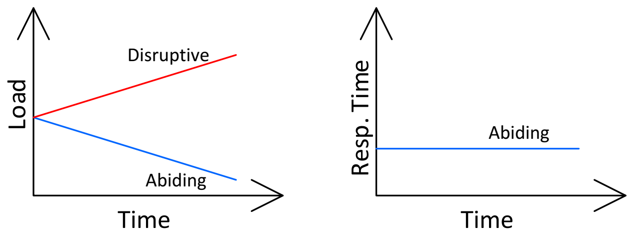


Fig. 16.2: Influence of disruptive tenants when abiding tenants adapt their workload accordingly

In the following, the idea is described in more detail. We start by measuring the isolation behavior of a non-isolated system by continuously increasing the disruptive tenant workload W_d. In such a situation, $z_t(W)$ remains unaffected if the workload of the abiding tenants W_a is adjusted accordingly to compensate for the increase in the disruptive tenant workload.

The x-axis in Figure 16.3 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 *Non-Isolated* line depicts how W_a has to decrease in order to maintain the same QoS as in the beginning. In a non-isolated system this function decreases linearly; that is, for every additional unit of work added to the disruptive tenant workload, one has to remove the same amount from the abiding tenant workload. In a perfectly isolated system, the increased disruptive tenant workload W_d would have no influence on $z_t(W)$ for all $t \in A$. Thus, W_a would be constant in this case as reflected by the *Isolated* line in the figure. The *Isolated* and *Non-Isolated* lines provide exact upper and lower bounds, which correspond to a perfectly isolated and a non-isolated system, respectively. Figure 16.3 shows some important data points, which are described in Table 16.2.

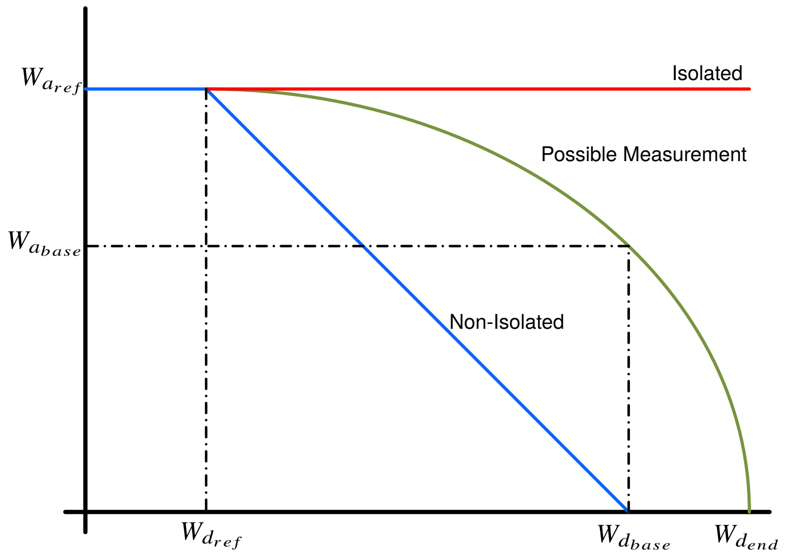


Fig. 16.3: Fictitious isolation curve including upper and lower bounds

Table 16.2: Description of relevant data points in Figure 16.3

Symbol	Meaning
W_d	The total workload induced by the disruptive tenants; $W_d = \sum_{t \in D} w_t$
$W_{d_{base}}$	The level of the disruptive tenant workload at which the abiding tenant workload in a non-isolated system must be reduced to 0 in order to avoid SLA violations
$W_{d_{end}}$	The level of the disruptive tenant workload at which the abiding tenant workload in the system under test (SUT) must be reduced to 0 in order to avoid SLA violations
$W_{d_{ref}}$	The value of the disruptive tenant workload at the reference point in the SUT with respect to which the degree of isolation is quantified; it is defined as the disruptive tenant workload at which, in a non-isolated system, the abiding tenant workload would have to start being reduced to avoid SLA violations
W_a	The total workload induced by the abiding tenants; $W_a = \sum_{t \in A} w_t$
$W_{a_{ref}}$	The value of the abiding tenant workload at the reference point $W_{d_{ref}}$ in the SUT; $W_{a_{ref}} = W_{d_{base}} - W_{d_{ref}}$
$W_{a_{base}}$	The value of the abiding tenant workload corresponding to $W_{d_{base}}$ in the SUT

Based on this approach, several metrics are defined in the following. As discussed before, the workload scenarios play an important role and it may therefore be necessary to consider multiple different scenarios.

16.2.3.1 Metrics Based on Edge Points

The edge points $W_{d_{end}}$, $W_{d_{base}}$, $W_{a_{ref}}$, and $W_{a_{base}}$ in Figure 16.3 provide several ways to define an isolation metric by themselves. I_{end} is a metric derived from the point at which the workloads of abiding tenants have to be reduced to 0 to compensate for the disruptive tenant workload. The metric describes a relationship between $W_{d_{end}}$ and $W_{a_{ref}}$. Due to the discussed relationship of the workloads in a non-isolated system and the definition of the various points, the condition $W_{a_{ref}} = W_{d_{base}} - W_{d_{ref}}$ holds. This relation helps to simplify the formulas. With Figure 16.3 in mind, the metric I_{end} is defined as follows:

$$I_{end} := \frac{W_{d_{end}} - W_{d_{base}}}{W_{a_{ref}}}. \tag{16.5}$$

A value of 0 for I_{end} reflects a non-isolated system. Higher values reflect better isolated systems. A value of 1 is interpreted as being twice as good as a non-isolated system. In case of a perfectly isolated system, the metric value tends to infinity. This makes it hard to interpret the value of the metric for a given system. A negative value may occur if, for some reason, the performance of the abiding tenants is reduced more than the disruptive tenant workload is increased. This may happen in case the system runs into an overloaded and trashing state.

Another approach to define an isolation metric uses $W_{a_{base}}$ as a reference. Setting this value and $W_{a_{ref}}$ in relation results in the following isolation metric having a value in the interval [0, 1]:

$$I_{base} := \frac{W_{a_{base}}}{W_{a_{ref}}}. \tag{16.6}$$

A value of 0 for I_{base} reflects a non-isolated system, while a value of 1 corresponds to perfect isolation. Both metrics have some drawbacks resulting from the fact that they do not take the curve's form into account. Consider a system that behaves like a perfectly isolated system until a short distance from $W_{d_{base}}$ and then suddenly drops to $W_a = 0$. In such a system, both metrics would have the same value as for a completely non-isolated system, which obviously is unfair in this case. Moreover, a well-isolated system requires a very high disruptive tenant workload before W_a drops to 0, which makes 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 between $W_{d_{ref}}$ and $W_{d_{base}}$. Given that the metric does not reflect what happens after $W_{d_{base}}$, it may lead to misleading results in the case of well-isolated systems for which the respective $W_{d_{end}}$ points differ significantly.

For systems that exhibit a linear degradation of the abiding tenant workload, it is also possible to use isolation metrics based on the angle between the observed abiding tenant workload's line segment and the line segment representing a non-isolated system. However, typically, a linear behavior cannot be assumed.

16.2.3.2 Metrics Based on Integrals

Next, we define two 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 $A_{non-isolated}$. The area under the curve corresponding to a non-isolated system is calculated as $W_{a_{ref}}^2/2$.

Integral Limited to $\mathbf{W_{d_{base}}}$ The first metric $I_{intBase}$ represents the isolation as the ratio of $A_{measured}$ and $A_{non-isolated}$ within the interval $[W_{d_{ref}}, W_{d_{base}}]$. $f_m : W_d \rightarrow W_a$ is defined as a function that returns the residual workload for the abiding tenants based on the workload of the disruptive tenants. Based on this function, we define the metric $I_{intBase}$ as follows:

$$I_{intBase} := \frac{\left(\int\limits_{W_{d_{ref}}}^{W_{d_{base}}} f_m(W_d)dW_d\right) - W_{a_{ref}}^2/2}{W_{a_{ref}}^2/2}. \tag{16.7}$$

$I_{intBase}$ has a value of 0 in case 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 metric's major advantage is that it helps to set the system directly in relation to an isolated and non-isolated system. This metric, again, has the drawback that it only captures the system behavior within $[W_{d_{ref}}, W_{d_{base}}]$. Again, a negative value may occur if, for some reason, the performance of the abiding tenants is reduced to a greater degree than the disruptive tenant workload is increased.

Integral Without Predefined Intervals In a well-isolated system, it would be of interest to measure the system behavior beyond the point $W_{d_{base}}$. The following metric $I_{intFree}$ allows the use of any predefined artificial upper bound $p_{end} > W_{d_{base}}$ representing the highest value of W_d that was measured in the SUT. The metric is defined as follows:

$$I_{intFree} := \frac{\left(\int\limits_{W_{d_{ref}}}^{p_{end}} f_m(W_d)dW_d\right) - W_{a_{ref}}^2/2}{W_{a_{ref}} \cdot (p_{end} - W_{d_{ref}}) - W_{a_{ref}}^2/2}. \tag{16.8}$$

This metric quantifies the degree of isolation provided by the system for a specified maximum level of injected disruptive tenant workload p_{end}. A value of 1 represents a perfect isolation; a value of 0 represents a non-isolated system. Negative values for $I_{intFree}$ have the same interpretation as negative values for $I_{intBase}$.

16.2.4 Further Isolation Quality Aspects

Although the metrics described in Sections 16.2.2 and 16.2.3 allow one to quantify isolation, they do not adequately describe the behavior of a system over time. Several methods for performance isolation employ an adaptive approach that dynamically adapts the system configuration to ensure isolation often based on a closed control loop—see, for example, Krebs, Spinner, et al. (2014). Consequently, one can assume the existence of situations where the system requires a certain amount of time to adapt to changes in the workload. Therefore, two additional metrics allowing one to quantify the dynamic aspects of performance isolation mechanisms are discussed.

Some commonly discussed issues in the context of system control theory in the literature are *stability/oscillation*, *settling time/performance*, and *accuracy/steady-state error* (Janert, 2013, pp. 19–21). In our context, the accuracy (steady-state error) is already covered by the metrics in Sections 16.2.2 and 16.2.3. The other two issues are discussed in the following two sections.

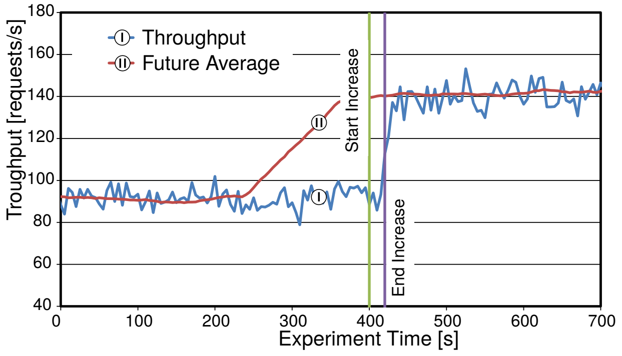
16.2.4.1 Settling Time

The settling time describes the time a system needs to achieve an output value within a defined error range after a sudden change in input levels. A system with a faster settling time is generally considered to be better.

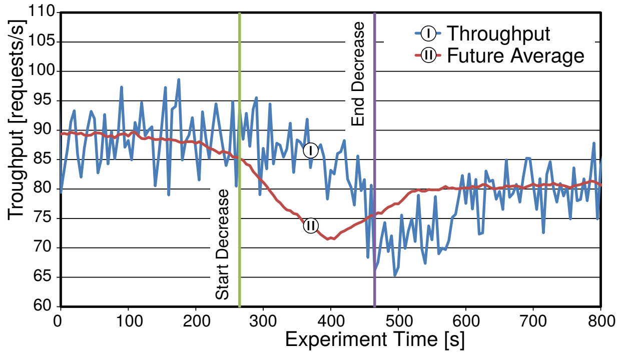
Ideally, a Dirac impulse would be used for the input. In our context, the input value is the workload of the tenants, whereas the output value is the observed value of the QoS metric under investigation (e.g., response time). Naturally, it is not possible to generate a Dirac impulse for such a system; therefore, a step function must be used. However, a significant increase of the workload to a constant value in a very short time may not be feasible. Therefore, the start event for measuring the settling time is defined as the point in time at which the workload again achieves stability. An observation of the QoS metric reaching a stable value can then be used as the trigger to stop the measurement of the settling time. In these measurements, a certain error is acceptable. It is possible to relate the start and stop events to the QoS guarantee provided to a tenant. In this case, the start event is triggered if the observed QoS is worse than the guarantee, whereas the end event is triggered when it meets the guarantee again.

However, a different approach would be required should the considered QoS metric not be related to any QoS guarantees. The proposed metric considers the average response time of the sample of next m to n observations in the future and compares it with the current one. The values of m and n should be selected in a way to fulfill the following conditions: (1) there should be enough sample data in the floating window to compute a stable average value; (2) m should be far enough in the future to ensure that the average value is already stable before the impulse is triggered; and (3) in cases of an online calculation, m should not be too far in the future in order to be able to obtain a timely result for the metric.[1] If the load increases, one can expect a higher response time, which will decrease as the method tries to compensate this problem. At some point in time, this value will be close to the computed average or even cross this line, which marks the end event.

Figure 16.4a,b shows an example throughput over time for an abiding tenant. The two vertical lines mark the beginning and the end of the time span where the workload changed. Note that the workload-related lines are based on the amount of simulated users in the benchmark, and it takes a few seconds to start them. The throughput itself needs even longer before adapting to the new workload.



(a) Load increased



(b) Load decreased

Fig. 16.4: Examples of measuring settling times

[1] Note that in the case of offline analysis with just one single impulse (e.g., benchmarking), the selection of m and n is less important and the threshold may even be computed by a separate measurement run.

Metrics similar to settling time were already used in the past for adaptive IT systems in the context of QoS metrics. One example is the CloudScale consortium (Brataas, 2014), which uses a metric referred to as MTTQR to describe the time an elastic system needs to become SLA-compliant after the occurrence of an SLA violation. Although MTTQR focuses on different scenarios, it is comparable to the interpretation of settling time presented here.

16.2.4.2 Oscillation

Oscillation can happen if feedback from the system is used to adapt it to changing scenarios. Figure 16.5 shows an example of oscillating throughput for abiding tenants.

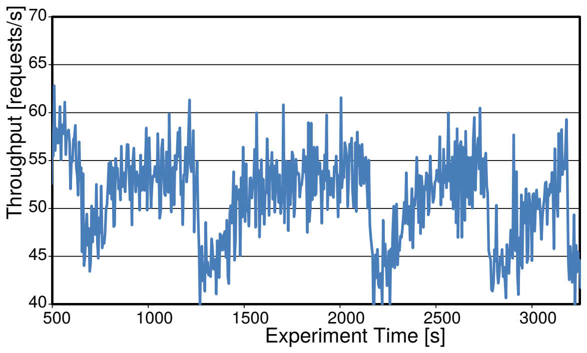


Fig. 16.5: Example of oscillation

Oscillation is the repetitive variation of the system between two or more different states. It is a common phenomenon in control theory (Janert, 2013, pp. 19–21). For this reason, controllers are usually designed to damp the oscillation and ensure that the amplitude converges to zero. If this is the case, the *settling time* is a useful metric. Otherwise, the controller maintains an unstable state.

Discrete systems with random inputs, like an interactive web application, can be in a steady state concerning the average values of QoS metrics, while the input is still subject to random processes. Furthermore, in closed systems, the output may influence the input. This increases the risk that the isolation method never converges to a steady state. The amplitude and the frequency of resulting oscillations are indicators to compare different methods. For the purpose of performance isolation mechanisms, the amplitude would be based on the average relative change of the QoS metric of interest. An average value for all tenants can be considered. Although this seems intuitively correct, such a metric would lack in objectivity. This is because in real systems, oscillation is mixed with noise in the measurements and a clear oscillation might not be visible at all. Furthermore, if the system reacts very fast to minor and possibly random changes, no repeatable pattern may occur. Thus, it is likely that a precise identification of the highest and lowest point of the oscillation is not possible. Consequently, it may be difficult to clearly identify patterns caused by the active control mechanisms as opposed to normal random processes. Thus, a human would have to define which signals are relevant and which are not, raising the question of objectivity and reliability of the metric. Furthermore, the distribution of the measured data would be unknown and potentially different for different isolation mechanisms.

Therefore, we consider the length of the interval between the 25% and 75% percentiles set in relation to the observed arithmetic mean or median value of the QoS metric of interest. In case of high oscillation or high variability of the metric, the length of the interval would be higher in comparison to scenarios with low oscillation. This approach does not rely on the assessment of a human. The drawback is that very strong noise may be classified wrongly as oscillation. This metric is closely related to an existing approach for quantifying performance variability (Iosup, Yigitbasi, et al., 2011).

16.3 Case Study

We now present a case study—the initial version of which was published in Krebs et al. (2014)—showing the metrics in action by applying them to virtualization-based systems. The case study demonstrates the applicability of the metrics in real-life environments and provides some insights on the isolation capabilities of the widely used hypervisor Xen. Furthermore, it is an example of how the metrics can be employed by system operators to make decisions in a deployment scenario.

Beside multi-tenancy, the sharing of hardware resources by running several operating systems on the same physical host is a widely adopted technology providing the foundation for IaaS clouds. Xen[2] is a widely used hypervisor for Linux environments enabling resource sharing at the hardware level. The goal of the case study we present in this section is to stress Xen in order to evaluate its performance isolation capabilities. More precisely, we quantify the degree of isolation for various Xen configurations and deployments based on a black-box measurement approach employing the isolation metrics introduced in the previous section. We deploy several instances of the TPC-W benchmark on different virtual machines (VMs) hosted by one Xen hypervisor and measure how they influence each other. The case study demonstrates the wide range of scenarios supported by the metrics and how the latter can be used to reason about the isolation capabilities of IaaS clouds running on Xen.

In the following, we describe some details on the Xen hypervisor, the chosen benchmark, and the system landscape we consider in our case study. We then present and discuss the evaluation results.

[2] Xen hypervisor: https://xenproject.org

16.3.1 Xen Hypervisor

A hypervisor is a software enabling the execution of several virtual machines (VM) as guests on one physical host. Xen is one of the most popular hypervisors for Linux environments. The operating systems installed within VMs are decoupled from each other and have no permission for administrative tasks on the hardware or the hypervisor's configuration. In order to configure the hypervisor and to execute administrative tasks, the first VM started in Xen (referred to as *domain-0* or *dom0*) has special privileges. Furthermore, dom0 provides a driver abstraction for the different guest systems. The drivers in Xen are divided into two parts. The drivers actually accessing the hardware are installed in dom0; the guest systems (referred to as *domU* domains) communicate with 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; thus, the independent guest VMs are likely to influence each other when executing such tasks. Such a behavior was observed by Huber et al. (2011) and Gupta et al. (2006). By default, VMs have access to all existing resources on the host. To increase performance and isolation, it is possible to pin a core exclusively to a domain. It is worth mentioning that dom0 usually does not host any services for end users due to its special administrative role.

16.3.2 TPC-W Benchmark

The TPC-W benchmark was introduced in Chapter 9, Section 9.3.3. In the specific setup of this chapter, TPC-W's bookshop consists of a Java Servlet-based application and an SQL database. Instead of using the usual performance metric, which is the number of web interactions processed per second, we consider the average response time of the requests for TPC-W's three profiles. The load can be varied by the amount of emulated browsers (EB) accessing the 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 the influence of the different VMs on each other.

16.3.3 Experimental Environment

The experimental environment in our case study comprises two servers with two physical quad core CPUs (2,133 MHz with two threads per core) and 16 GB of main memory. On both servers, Xen 4.1 is installed and Suse Linux Enterprise Server (SLES) 11 SP2 is used as a guest operating system. The servers are connected with a 1 Gbit Ethernet link. One server hosts the load driver for the TPC-W benchmark. The various domains of the second server are described below as part of the scenario-specific configuration. The database schema is refreshed before every measurement and filled with 100,000 items and 300,000 customers.

In total, we study three different configuration scenarios in our case study. In the *pinned* scenario, the server hosts four guest systems (dom1, dom2, dom3, dom4) and dom0. Every domU domain has a fixed memory allocation of 3,096 MB and hosts a MySQL 5.0 database and an SAP-specific customized Tomcat web server. The various domains were pinned exclusively to the existing cores. Thus, no competition for the same CPU resources was possible. Based on this run-time environment, four separate instances of the TPC-W bookshop application were deployed.

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

In addition to this, we investigated an *unpinned two-tier* scenario, which also does not have a fixed CPU pinning and likewise uses the Xen credit scheduler. However, the database and the application server in this case were deployed in separate domains. Every domU domain with an application server has a fixed memory allocation of 2,024 MB and the database domain uses 1,024 MB. This memory setup was chosen because of the small database size.

Table 16.3 shows the values we used to define the reference and disruptive tenant workloads for the three scenarios. The number of emulated browsers (EBs) at the maximum aggregated throughput of all domains is presented in the second column; the corresponding throughput per domain and the average response time are listed next. The last column shows the disruptive domain's 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-impact-based and workload-ratio-based metrics. The relevant QoS for our analysis is the average response time of the tenants. The additional information is shown only for the sake of better system understanding.

Table 16.3: Scenario setup and configuration

Scenario	EBs per domU	Total throughput	Throughput per domU	Avg. resp. time	Max. load disruptive
Pinned	3,000	1,195 r/s	299	1,104 ms	15,000
Unpinned	1,600	721 r/s	180	843 ms	13,500
Unpinned two-tier	1,300	617 r/s	154	833 ms	8,000 (QoS-based), 11,050 (ratio-based)

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

As a consequence of these observations p_{end} is set to 15,000 for the pinned scenario and to 13,500 for the unpinned. In the unpinned two-tier scenario, we had to set p_{end} to 11,050 and stop our test for the I_{QoS} metrics at 8,000 users. It is worth mentioning that in both unpinned scenarios, p_{end} is very close to nine times the load of the maximum throughput for one domain.

In all presented examples, one tenant has been used to generate the disruptive load. All other tenants have been classified as abiding tenants.

16.3.4 Performance Isolation Metrics in Action

We now provide an overview of the measurement results and the observed isolation metrics. Figure 16.6 combines the results for both unpinned scenarios based on normalized values for the abiding and disruptive tenant workloads. Table 16.4 presents the QoS-impact-based metrics based on the same values for Δw. Thus, the results provide a comparable view for the two deployments.

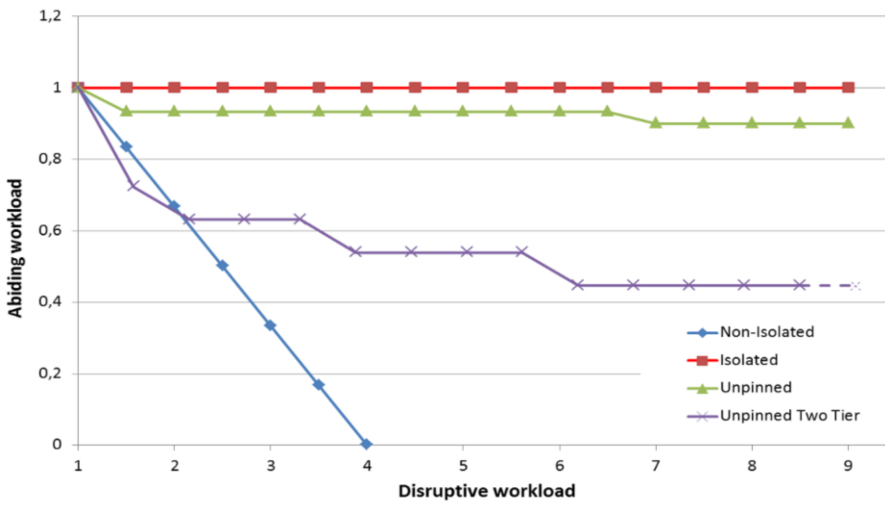


Fig. 16.6: Normalized reduction of abiding tenant workload in the unpinned and the unpinned two-tier scenario

Table 16.4 contains the values of I_{QoS} for all three scenarios. The first column of Table 16.4 shows the scenario, the second column shows the number of users in the disruptive domain, and the third column shows the average response time of all abiding domains followed by the results for Δw, Δz, I_{QoS}, and I_{avg}. For the pinned scenario, we collected only one measurement due to the very good isolation. The I_{avg} values were calculated based on interpolation of the depicted measurements in the table as supporting points including the reference workload ($\Delta w = 0$ with $\Delta z = 0$).

For the pinned scenario, we assume a behavior of the isolation between $\Delta w = 0$ and $\Delta w = 1$.

Table 16.4: Results of I_{QoS} in the different scenarios

Scenario	Disruptive load	Response time	Δw	Δz	I_{QoS}	I_{avg}
Pinned	15,000	1,317 ms	1.00	0.19	0.19	0.10
Unpinned	3,200	927 ms	0.25	0.10	0.40	0.21
	4,800	942 ms	0.50	0.12	0.24	
	7,500	914 ms	0.92	0.09	0.09	
	10,000	1,173 ms	1.31	0.39	0.30	
Unpinned two-tier	3,000	1,011 ms	0.33	0.21	0.64	3.06
	4,400	3,784 ms	0.60	3.54	5.90	
	6,750	4,354 ms	1.05	4.22	4.02	

16.3.4.1 Pinned Scenario

Overall, this scenario presented a nearly perfect isolation throughout the whole range. The I_{QoS} metric presented in Table 16.4 at a disruptive tenant workload of 15,000 users was below 0.2 and the I_{avg} resulted in 0.1. The workload-ratio-based metric decreased for the abiding tenant workload only once at 12,000 disruptive tenant users. The related metrics $I_{intFree15000}$ and $I_{intBase}$ resulted in a value slightly below 1.

16.3.4.2 Unpinned Scenario

For the metrics based on the QoS impact, we determined the isolation at various disruptive tenant workloads shown in Table 16.4. We observed two significant characteristics. The first one is the increasing response time when the disruptive tenant workload is set to 3,200 users. The second one is the increasing response time at 10,000 users. Accordingly, the isolation becomes better between 3,200 users and 10,000 users. This is due to the widely stable response times at increasing load, which changes the ratio of $\Delta z/\Delta w$. On average, the isolation I_{avg} is 0.21.

Figure 16.6 presents the total abiding tenant workload W_a based on the disruptive tenant 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 of the abiding tenants. At a disruptive tenant workload of 13,500 users (corresponding to 9 in the figure), the disruptive domain failed to successfully handle incoming requests. Therefore, the results are not valid for higher disruptive tenant workloads. The overall isolation values are $I_{intFree13500} = 0.89$ and $I_{intBase} = 0.86$.

16.3.4.3 Unpinned Two-Tier Scenario

Table 16.4 shows the various disruptive tenant workloads used to evaluate I_{QoS}. We configured the disruptive tenant workloads in a way to result in the same Δw as in the unpinned single-tier scenarios. Due to the increasing number of timeouts and exceptions in the disruptive domain, we had to stop at 6,750 users. For this workload range, we observed continuously increasing response times. Nevertheless, from 4,400 to 6,750 users, the isolation improved, as Δw increased more than Δz. Over the entire range of measurements, the average isolation I_{avg} was 3.06.

Figure 16.6 presents the total abiding tenant workload W_a based on the disruptive tenant users for the workload-ratio-based metrics. Analogous to the response times in Table 16.4, we can see a continuously decreasing amount of abiding tenant workload in Figure 16.6. At a disruptive tenant workload of 2, we can see the observed isolation curve crossing the respective curve for a non-isolated system. This is due to the selected step width for reducing the number of users in the disruptive domain. At a disruptive tenant workload of 11,050 users (corresponding to 8.5 in Figure 16.6), the disruptive domain failed to successfully handle incoming requests. The results are no longer valid for higher disruptive tenant workloads and are therefore illustrated using a dashed line. The overall isolation values were $I_{intFree1105} = 0.42$ and $I_{intBase} = 0.36$.

16.3.5 Effectiveness of the Deployment Options

Overall, the *pinned* scenario exhibited the best results, whereas the *unpinned two-tier* scenario exhibited the worst ones. The selected size of the database was small enough for data to be mostly cached. The memory was not overcommitted in our setup and the network I/O did not reach the 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 domain were shared with those for dom0; therefore, the slightly increased I/O overhead for dom0 was competing for resources and had some minor effect. The credit scheduler was not able to compensate completely for this. By splitting the domU domain into application server and database server, we noticeably increased the network I/O. In this setup, we observed a significant impact of the disruptive domain on the others, whereby the handling of the network I/O in dom0 led to a bottleneck and/or it 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 and thus this setup might lack in terms of efficiency. The *unpinned* scenario overcomes this drawback at the expense of performance and isolation. From a separation-of-concerns point of view, it might be beneficial to separate the database and application server. On the other hand, as can be seen from Table 16.3, a distributed deployment provides less performance and the worst isolation. These example measurements show how the isolation metrics provide the opportunity to quantify one more dimension in the framework of multiple trade-off decisions a system provider has to make. An additional result is that an administrator can increase isolation by hard resource allocations, which also lead to reduced I/O.

16.4 Assessment of the Metrics

For the assessment of the metrics, we concentrate on the following aspects: First, the practical usability of the metrics for the target group of system owners/providers or developers/researchers; and second, the expressiveness of the metrics in terms of the type of evidence they provide; third, the number of measurements required to obtain a valid value; fourth, situations in which the metrics are not meaningful. In the following, we evaluate the metrics of each category with respect to these aspects.

16.4.1 QoS-Impact-Based Metrics

These metrics show the influence of disruptive tenant workloads on the QoS of abiding tenants. This helps system owners to manage their systems, because it indicates the influence of disruptive tenant workloads on the QoS, which is important for capacity planning. QoS-impact-based metrics can show that a system is perfectly isolated; however, they fail in ranking a system's isolation capabilities in the range between perfectly isolated and non-isolated. Thus, it is hard to estimate the potential of an isolation method. 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 on the system's performance isolation capabilities, more measurements are required. Therefore, I_{avg} describes the average isolation value for multiple different scenarios of interest. Nevertheless, the metric is not suitable to describe a system's impact of different disruptive tenant workloads on the abiding tenants, because these workloads cannot be set into relation for a concrete scenario.

16.4.2 Edge-Point-Based Metrics

The metric I_{end} might not be practically usable for quantifying isolation in well-isolated systems. Furthermore, it is not possible to directly deduce from it relevant system behaviors such as response time behavior. If this metric is provided, it could help to compare two systems regarding the maximum disruptive tenant workload they can handle. However, to quantify I_{end}, more measurements are required than would be the case for the QoS-impact-based metrics.

I_{base} orders a system within the range of perfectly 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 also inadequate to derive direct QoS-related metrics. The usefulness of this metric is limited compared to the integral-based metrics.

16.4.3 Integral-Based 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 additionally specified in order to have a fully defined metric. Both metrics provide good evidence of the isolation within the considered interval ordered between the magnitudes of perfectly isolated and non-isolated systems. However, they lack in providing information concerning the degree of SLA violations. For example, the SLA violations could be very low and acceptable or critically high in each iteration as we reduce W_a. However, in both cases, the results of the metrics would be similar. This limits the value of $I_{intBase}$ and $I_{intFree}$ for system owners/providers. Nevertheless, 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 h.

16.4.4 Discussion

The various metrics show their advantages in different fields of applications and express various semantics. The I_{QoS} and I_{avg} metrics capture the reduced QoS due to disruptive tenant workload. They cannot provide a ranking within the range of fully isolated and non-isolated systems. However, for a system operator this might be helpful to estimate the impact of disruptive tenant workloads on the system. The I_{end} metric shows how many times a system is better than a non-isolated one. This information may be helpful to compare different systems if one has to decide for one. The integral-based metrics rank a system within the range of fully isolated and non-isolated. This knowledge is beneficial for the developer of a system to estimate the potential for improvements.

The presented isolation metrics are not limited to multi-tenant environments. They are also applicable in other scenarios where a system is shared, for example, a web service triggered by other components, virtual machines hosted on the same hypervisor instance (as shown in our case study), or network devices serving packets from various sources. However, practical limitations might appear, for example, due to non-uniform workload behavior with work arriving from different sources.

16.5 Concluding Remarks

This chapter presented metrics to quantify the degree of performance isolation a system provides. The metrics are applicable for use in performance benchmarks that measure the performance without requiring internal knowledge. They are preferable in situations where different request sources use the functions of a shared system with similar demands per request but with a different load intensity. These characteristics are typical for multi-tenant applications but can also occur in other shared resource systems. The presented metrics are based on observing the influence of disruptive tenants (i.e., tenants exceeding their assigned quotas) on the abiding tenants (i.e., tenants working within their quotas). Thus, the influence on one group as a function of the workload of the other group must be evaluated. This is a major difference to traditional performance benchmarking. We presented a case study showing the metrics in action by applying them to evaluate the performance isolation of virtual machines running on a shared physical host. The case study demonstrated the applicability of the metrics in real-life environments and provided some insights on the isolation capabilities of the widely used virtualization platform Xen. Furthermore, it showed an example of how the metrics can be employed by system operators to make decisions in a deployment scenario. The performance isolation metrics presented in this chapter can be applied to quantify isolation with respect to any measurable QoS-related property of a system shared between different entities.

References

Brataas, G. (2014). *CloudScale: Design Support Deliverable D1.2*. EU FP7, Collaboration Project CloudScale. FP7-ICT-2011-8-317704 (cited on p. 353).

Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). “Benchmarking Cloud Serving Systems with YCSB”. In: *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC 2010)*. (Indianapolis, Indiana, USA). ACM: New York, NY, USA, pp. 143–154 (cited on p. 343).

Gupta, D., Cherkasova, L., Gardner, R., and Vahdat, A. (2006). “Enforcing Performance Isolation Across Virtual Machines in Xen”. In: *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*. (Melbourne, Australia). Springer US: New York, NY, USA, pp. 342–362 (cited on p. 355).

Herbst, N. R., Bauer, A., Kounev, S., Oikonomou, G., Eyk, E. van, Kousiouris, G., Evangelinou, A., Krebs, R., Brecht, T., Abad, C. L., and Iosup, A. (2018).

"Quantifying Cloud Performance and Dependability: Taxonomy, Metric Design, and Emerging Challenges". *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 3(4). ACM: New York, NY, USA, 19:1–19:36 (cited on p. 342).

Herbst, N. R., Kounev, S., and Reussner, R. (2013). "Elasticity in Cloud Computing: What it is, and What it is Not". In: *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013)*. (San Jose, CA). USENIX (cited on p. 343).

Herbst, N. R., Krebs, R., Oikonomou, G., Kousiouris, G., Evangelinou, A., Iosup, A., and Kounev, S. (2016). *Ready for Rain? A View from SPEC Research on the Future of Cloud Metrics*. Tech. rep. SPEC-RG-2016-01. Gainesville, VA, USA: SPEC RG—Cloud Working Group, Standard Performance Evaluation Corporation (SPEC) (cited on p. 342).

Huber, N., von Quast, M., Hauck, M., and Kounev, S. (2011). "Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments". In: *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011)*. (Noordwijkerhout, The Netherlands). SciTePress, pp. 563–573 (cited on p. 355).

Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. (2011). "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing". *IEEE Transactions on Parallel and Distributed Systems*, 22(6). IEEE: Piscataway, New Jersey, USA, pp. 931–945 (cited on p. 343).

Iosup, A., Yigitbasi, N., and Epema, D. (2011). "On the Performance Variability of Production Cloud Services". In: *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011)*. (Newport Beach, CA, USA). IEEE: Piscataway, New Jersey, USA, pp. 104–113 (cited on p. 354).

Islam, S., Lee, K., Fekete, A., and Liu, A. (2012). "How a Consumer Can Measure Elasticity for Cloud Platforms". In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. (Boston, MA, USA). ACM: New York, NY, USA, pp. 85–96 (cited on p. 343).

Janert, P. (2013). *Feedback Control for Computer Systems*. O'Reilly and Associates: Sebastopol, California (cited on pp. 351, 353).

Krebs, R. (2015). "Performance Isolation in Multi-Tenant Applications". PhD thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT) (cited on p. 342).

Krebs, R., Momm, C., and Kounev, S. (2012). "Architectural Concerns in Multi-Tenant SaaS Applications". In: *Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*. (Setubal, Portugal). SciTePress (cited on p. 341).

– (2014). "Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments". *Science of Computer Programming*, Vol. 90, Part B. Elsevier Science: Amsterdam, The Netherlands, pp. 116–134 (cited on pp. 342, 354).

Krebs, R., Spinner, S., Ahmed, N., and Kounev, S. (2014). “Resource Usage Control In Multi-Tenant Applications”. In: *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014)*. (Chicago, IL, USA). IEEE: Piscataway, New Jersey, USA, pp. 122–131 (cited on p. 351).

Kupperberg, M., Herbst, N. R., Kistowski, J. von, and Reussner, R. (2011). *Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms*. Tech. rep. 2011-16. Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT) (cited on p. 343).

Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. (2010). “Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance”. *Proceedings of the VLDB Endowment*, 3(1-2). VLDB Endowment, pp. 460–471 (cited on p. 343).