QoS-BASED RESOURCE ALLOCATION FRAMEWORK FOR MULTI-DOMAIN SLA MANAGEMENT IN CLOUDS
Kuan Lu, Ramin YahyapOUR, Philipp Wieder, Constantinos Kotsokalis, Edwin Yaqub, Ali Imran Jehangiri
 Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
 IT & Media Center of Dortmund University of Technology, Germany
 {kuan.lu, ramin.yahyapour, philipp.wieder, edwinyaqub, ali.jehangiri}@gwdg.de
 constantinos.kotsokalis@tu-dortmund.de

Abstract
In clouds, current virtualization technologies of IaaS enable the live migration of running VMs to achieve load balancing, fault-tolerance and hardware consolidation in data centers. However, the downtime / service unavailability due to live migration may be substantial with relevance to the customers’ expectations on responsiveness, as the latter are declared in established SLAs, which define all relevant aspects of the services between service provider and customer. Moreover, the service unavailability may cause significant (potentially exponential) SLA violation penalties to its associated higher-level domains (e.g., PaaS and SaaS). Therefore, in order to deliver high availability service, VM live migration should be arranged and managed carefully. In this paper, we present the OpenStack version of Generic SLA Manager, alongside its strategies for VM selection and allocation during live migration of VMs. Based on the proposed autonomous SLA violation-filtering framework, we simulate a use case where IaaS (OpenStack-SLAM) and PaaS (OpenShift) are combined; and assess performance and efficiency of the aforementioned VM placement strategies, when a multi-domain SLA pricing & penalty model is involved. We find that our proposal is efficient in managing trade-offs between the operational objectives of service providers (including financial considerations) and the customers’ expected QoS requirements.

Keywords: Live migration, Virtual machines, IaaS, PaaS, Availability, SLA pricing, SLA penalties, SLA violation, Resource allocation

1. INTRODUCTION
In Infrastructure-as-a-Service (IaaS), through virtualization technologies (e.g., VMWare (2013), Xen (2013)), physical resources of data centers can be partitioned into flexible and scalable virtual computing units, namely Virtual Machines (VMs). However, large-scale data centers introduce large power consumption costs. Thus, an efficient technique that dynamically reconfigures the IT infrastructure to reduce the total power consumption becomes necessary. As such, VM consolidation emerges to execute the VMs on as few servers as possible, to concentrate the workloads and limit the number of physical servers powered on.

VM consolidation is usually treated as an objective of the Service Provider (SP). From customer’s perspective, an automated negotiation may be used to accommodate heterogeneous requirements against an SP’s capabilities and acceptable usage terms. The result of such a negotiation is a Service Level Agreement (SLA), an electronic contract that establishes all relevant aspects of the service. During the SLA negotiation, all terms must be evaluated before a final agreement is reached. In order to commit to the requested Quality-of-Service (QoS) terms (e.g., service performance and availability), SPs have to assess their own resource management strategies as to trade-off profit making with a guaranteed delivery of service(s) and avoid penalties in case the agreement is violated at runtime. An aggressive consolidation of VMs however may lead to performance degradation when the service faces increasing demand, resulting in unexpected rise of resource utilization.

By means of VM live migration technologies, both VM consolidation and service performance can be coordinated and balanced. However, research by Salfner et al. (2012) revealed that short downtimes of services are unavoidable during VM live migration due to the overheads of moving the running VMs. Hence, the respective service interruptions in IaaS reduce the overall service availability and it is possible that the customer’s expectations on responsiveness are not met. Moreover, it might bring exponential service violation penalties to its associated domains (e.g., Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS)). By way of an example, the solutions from Beloglazov et al. (2011) provide service availability from 99.70% to 99.93%. According to the research by Akoush et al. (2010), for e-commerce and other industrial use cases, an availability value below 99.90% is usually considered unacceptable. In order to provide high availability, to avoid service violation and the subsequent penalties, the number of VM live migrations should be monitored and controlled.

In this paper, we present an OpenStack version of the Generic SLA Manager (GSLAM), which could be potentially used into project PaaSage (2012). We apply this software system to combine IaaS (OpenStack-SLAM) and PaaS (OpenShift (2013)) in a use case that features multi-domain SLA management. Via the introduction of a pricing
and penalty model that considers such multi-domain scenarios, we apply our resource allocation strategies for VM selection and allocation during live migration. Based on the proposed autonomous SLA violation-filtering framework, we simulate the full scenario (using the CloudSim platform from Calheiros et al. (2011)) and illustrate the suitability of our proposal for the efficient management of VM live migration. Thereby, the agreed service availability is not violated without paying the extra penalties and a trade-off between the SP’s objectives and the customers’ expected QoS requirements can also be achieved successfully.

The remainder of this paper is structured as follows. In Section 2, we will discuss the related work. Section 3 will present the OpenStack version of the GSLAM. In Section 4, a formal model of SLA pricing and penalty will be provided. Section 5 introduces our autonomous SLA violation-filtering strategy. In Section 6, a description of our approach to process resource allocation and how it achieves important resource management objectives will be given. Then, in Section 7, we will validate our mechanisms performing discrete event simulations. Finally, we conclude the paper in Section 8.

2. RELATED WORK

Many publications, such as Kertesz et al. (2008), Stantchev and Schröpfer (2009) and Brandic et al. (2009), discuss the topic of SLA management for IT clouds, but most of them are looking at it from a conceptual and architectural point of view. To the best of our knowledge, there is no prior work for SLA management in OpenStack that considers multi-domain pricing and penalties, and that can be further applied to develop QoS-aware resource allocation strategies.

Many works, such as Kosinski et al. (2008), Becker et al. (2008) and Rana et al. (2008), also discuss SLA violations. However, there are few to model the consequences of the violations, namely, SLA penalties. Currently, although some approaches describe penalties, they do not satisfy all of the following requirements for formulating complex penalty expressions in a single unambiguous model:

- Able to present the chain effect on violation penalties among multi-domains in IaaS.
- Full flexibility regarding QoS levels agreed and/or achieved, without being constrained (e.g., by pre-specified classes of service).
- Openness and applicability to different domains, without dependence on specific languages or taxonomies.

As regards VM live migration, there are certain approaches already widely utilized. Through shared storage such as iSCSI Storage Area Network (SAN) or Network Attached Storage (NAS), the process of live migration is reduced to only copying memory state and CPU registers from source host to destination, without transferring the whole VM. In contrast to the pure stop-and-copy strategy of offline VM migration, live migration fine-tunes the migration into several rounds of pre-copying and one last round of stop-and-copy with much less downtime. Nevertheless, based on the impact of VM migration from Hermenier (2009), there are still a few issues to address. Firstly, the more iteration in the pre-copying phase, the less data need to be transferred in the final stop-and-copy phase, and the downtime becomes less. However, as Akoush et al. (2010) explained that the short downtime comes at the cost of longer total migration time, which leads to significant influence on service performance. Secondly, the downtime will never be eliminated, because many workloads usually include some memory pages that are updated frequently, named Writeable Working Sets (WWS). Clearly, it is wise not to maintain it until the last phase. Moreover, for stop-and-copy, Salfner et al. (2012) noted that the performance of VM live migration is affected by many factors, such as workload type, hypervisor type and so on.

Thus, no visible downtime is only an ideal goal. The number of migrations should be controlled, as it has impact on the performance of the other running VMs in the source and destination hosts, which is mainly proportional to the amount of memory allocated to the VM. In this paper, the VM performance violation happens when the VM experiences the CPU utilization of 100%. And the performance degradation happens during the VM live migration.

Service availability is one of the most important QoS metrics in IT clouds. Machado and Stillerm (2011) outlined that the service availability guaranteed by three large cloud providers (Amazon, Google and Rackspace Cloud) is more than 99.90% in order to obtain good reputation in today’s competitive market. Therefore, in upcoming sections we propose to provide a basic service availability of 99.90% and an advanced availability of 99.99% in the scope of VM live migration. The later one implies that the SP has to pay special attention (e.g., extra resources) on the service in order to avoid SLA violations.

For VM consolidation, Beloglazov et al. (2012), Beloglazov and Rajkumar (2011) and Corradi et al. (2012) mainly discussed:

- Resource management, either in simulation or some prototype implementations within the common cloud middleware. And no SLA management is introduced.
- Using either artificial workloads or partial historical information from the workloads in various projects.
- Using VM live migration to leverage the consolidation and service performance; however, the migration was misused without carefully taking service availability into consideration.

Therefore, in this paper, our goals are the following:
- A proof-of-concept prototype implementation of OpenStack SLA management (IaaS), which aims to be connected with PaaS and SaaS layers by providing SLA lifecycle, service customization and automatic scalability.
- By using the workloads information at GWDG, our strategies are simulated in CloudSim and compared with others in several aspects.
- The influence of SLA chain penalties in multi-domains is introduced into VM consolidation. Also, availability oriented VM allocation strategies that control VM live migration and leverage the objectives between the SP and the customer.

3. OpenStack SLA Management Framework

Based on the GSLAM in our previous project SLA@SOI (2009), an OpenStack SLAM is presented. The GSLAM (Figure 1) provides a generic architecture that can be used across different domains and use cases to manage the entire SLA life cycle, including activities such as SLAs modeling, negotiation, provisioning, resources outsourcing, monitoring and adjustment. Gonzalez et al. (2011) argued that the GSLAM is an orchestrator of the generic components. It could provide interoperability and separates the SLAMs from specific representations of an SLA or SLA template. Namely, any domain specific scenario is able to contact with GSLAM to request a new set of generic components, which encapsulate all the basic functionalities for handling SLAs. Moreover, the corresponding planning and optimization component (POC) and provisioning and adjustment component (PAC) will be partially re-implemented so as to enrich the functionalities subject to it own domain. Through OpenStack Nova API Woorea (2012), the GSLAM is able to implement its corresponding Infrastructure Service Manager (ISM) by:
- Querying the status of infrastructure during SLA negotiation, based on which the SP is able to generate the corresponding offer / counter-offer for the customer.
- Providing SLA terms (i.e. pricing, penalty, availability and performance) monitoring mechanism together with OpenStack.
- Deploying the VM allocation strategies within OpenStack Nova so as to maximize the profit and minimize SLA violation as well as energy consumption.
- Creating, customizing and deploying the agreed services.
- Reconfiguring and removing the services on demand.

PaaSage (2012) aims to deliver an open and integrated PaaS for different (e.g., industrial, e-science and public sector) use cases to support model-based lifecycle management of cloud applications. Using OpenShift (2013), one of the most popular PaaS implementations, it could auto-scale its cloud PaaS framework for Java, Perl, PHP, Python and delivered in a shared-hosting model. PaaS permits many applications offered by multiple development teams to co-exist on the same set of hosts in a safe and reliable fashion. In addition to that, the platform offers a variety of opportunities for multi-tenant deployments. Thus, an application that is intended to work for a single organizational unit can also be deployed in such a manner that many organizations or end-users are able to use it. Therefore, end-users benefit from application management (instead of VM level management) while application providers can bring their applications into the PaaS cloud with minimal effort.

Figure 1. Integration of the Generic SLA Manager and OpenStack

http://hipore.com/ijcc
As Figure 2 illustrates, OpenShift can be treated as a customer of the OpenStack-SLAM asking for infrastructure support. Our target is to automatically scale up and down the virtual resources (i.e. VMs) for the PaaS domain as needed. The SLAM not only provides the VMs, it is also able to customize the VMs using pre-defined scripts so as to deliver the “OpenShift-ready” VMs in one click. Specifically, let us suppose a PaaS SP starts SLA negotiation with an IaaS SP. When the IaaS SP has sufficient resources, a counter-offer will be sent back to the PaaS SP with a timeout. Once the offer is accepted within the timeout, then the VM will be created, and the SLAM will automatically log into the VM by matching the public key pair with its private key. Finally, the pre-installed scripts will be executed on the target VM. The execution includes three steps, namely:

- Installing the PaaS broker-specific packages.
- Assigning a public IP for the VM.
- Configuration of Mongo database / ActiveMQ / other components associated with the PaaS broker in OpenShift.

Thus, the VM can be recognized and controlled by the PaaS broker. Similarly, if the host is detected as under-loaded, the infrastructure can be easily scaled-down by removing the VMs. Here, PaaS and IaaS layers are technically interconnected. In Section 4, we will see how are they mutually influenced in term of economical aspect during SLA violation.

![Sequence diagram for the negotiation between PaaS and the OpenStack SLA Manager](image)

**Figure 2. Sequence diagram for the negotiation between PaaS and the OpenStack SLA Manager**

### 4. MODELING OF SLA PRICING AND PENALTY

As Lu et al. (2011) argued that IaaS SPs are able to compute the minimum implementation costs as part of price quotations towards customers, in order to remain competitive. At the same time, profit and SLA violation probability constraints are used to decide whether the problem can be satisfied at all, and what is the decision space based on which implementation costs can be calculated. Furthermore, outsourcing via subcontracts was included as part of the decision process, to achieve additional profit but also to sustain customers when local resources are not sufficient.

Here, we assume that the corresponding planning and optimization strategies implemented by Lu et al. (2011) are fully applied but not explained in detail in order to keep the paper reasonably concise. In brief, the objective of the approach is to minimize implementation and outsourcing costs for reasons of competitiveness, while respecting business policies for profit and risk. A greedy algorithm for outsourcing was implemented, using cost and subcontractor reputation as selection criteria; and local resource configurations as a constraint satisfaction problem for acceptable profit and failure risks. Thus, it becomes possible to provide educated price quotes to customers and establish safe electronic contracts automatically. Discarding either local resource provisioning, or outsourcing, models...
efficiently the specialized cases of infrastructure resellers and isolated infrastructure providers respectively.

Therefore, let us assume an IaaS service $i$, and an SLA that governs consumption of this service by a certain customer. We have:

$$C^i = C_i^{\text{impl}} + Pr^i$$  \hspace{1cm} (1)

$$C_i^{\text{impl}} = C_i^i + C_i^e$$  \hspace{1cm} (2)

$$C_i^i = C_{\text{energy}}^i + C_{\text{utility}}^i$$  \hspace{1cm} (3)

where the cost $C^i$ of service $i$ is the sum of internal cost (i.e. internally utilized resources, energy cost) and external cost (i.e. sub-contracted resources) as well as profit.

In PaaS, a container includes a set of resources that allows users to run their applications. By delivering such a computing platform (e.g., operating system, program execution environment), many containers can be run simultaneously on one VM (see Figure 3). We assume on each VM there are $n$ containers.

Therefore, the cost of each PaaS service $p$ is:

$$C^p = \frac{C^i}{n} + Pr^p$$  \hspace{1cm} (4)

SaaS developers can implement and deploy their applications on a cloud platform (e.g., container) without the cost and complexity of buying and managing the underlying hardware and software layers. Similarly, we assume on each container $m$ applications are allocated. Then, the cost of a SaaS service $s$ is defined as following:

$$C^s = \frac{C^p}{m} + Pr^s$$  \hspace{1cm} (5)

$C^i$ and $C^p$ apply only based on the assumption that the payment has no implementation costs other than the infrastructure and platform environment for service execution.

Meanwhile, an SLA should also contain a set of penalty clauses specifying the responsibilities in case the SPs fail to deliver the pre-agreed QoS terms. Thus, we will use a variation of penalty model from Kotsokalis et al. (2011) as outlined in Equation 6: service $s$ is defined as following:

$$R_s = \sum GW_k \cdot VR_k$$  \hspace{1cm} (6)

where $R_s$ is the penalty ratio associated with the cost of service $x$, where $GW_k$ is the weight of one specific guarantee being violated, for this specific combination of guarantees. This value may be arbitrarily high. It allows the negotiating customer to express the importance of honoring certain guarantees in this penalty function. $VR_k$ is the violation ratio: the relationship between achieved quality and planned quality. It indicates how far the offered quality has drifted from the agreed quality of a specific service parameter.

Therefore, the penalty of IaaS service $i$ is:

$$P^i_s (QoS_1^i, ..., QoS_t^i) = C^i \cdot R_s$$  \hspace{1cm} (7)

The penalty of all PaaS services $p$ is:

$$n \cdot P^p_s (QoS_1^p, ..., QoS_t^p) = n \cdot C^p \cdot R_p$$  \hspace{1cm} (8)

By applying Equation 4, we have:

$$n \cdot C^p \cdot R_p = (C^i + n \cdot Pr^i) \cdot R_p$$  \hspace{1cm} (9)

The penalty of all SaaS services $s$ is:

$$m \cdot n \cdot P^s_s (QoS_1^s, ..., QoS_t^s) = m \cdot n \cdot C^s \cdot R_s$$  \hspace{1cm} (10)
By applying Equation 5, we have:

\[
m \cdot n \cdot C^e \cdot R_e = (C^e + n \cdot Pr^e + n \cdot m \cdot Pr^e) \cdot R_e \quad (11)
\]

The violation of some QoS terms on the IaaS layer will automatically affect the other domains. For example, unavailability of a VM will unquestionably enforce its inner PaaS and SaaS services to be unavailable. For all these QoS terms in the three layers, we have \( R_i = R_p = R_s = R \). Thus, the extra penalties of the PaaS layer comparing with the IaaS layer is:

\[
(9) - (7) = n \cdot Pr^p \cdot r \quad (12)
\]

Similarly, the extra penalties of SaaS layer comparing with IaaS layer is:

\[
(11) - (7) = (n \cdot Pr^p + m \cdot n \cdot Pr^s) \cdot r \quad (13)
\]

Hence, a slight availability violation in IaaS will lead to exponential influences on its associated domains (PaaS and SaaS). An IaaS SP, in order to compliant with the SLAs, has to make optimal reaction and adjustment while the service is running.

5. AUTONOMOUS SLA VIOLATION MONITORING AND FILTERING

Violation can be seen as one of states during the whole lifecycle of an SLA (as Figure 4 illustrated). As mentioned in Section 3, negotiation is of fundamental interest in service-oriented systems. Two or more parties negotiate SLAs based on their individual goals. Specifically, a user may initiate the SLA negotiation by sending pre-customized SLA template (\( S_0 \)). An SLA template is a document that describes a set of options for each domain specific service. The negotiation may last several rounds (\( S_1 \)) to negotiate offers or counter-offers, and both of them may require certain iterations. At some point, a final agreement is reached (\( S_3 \)). The negotiation could also be terminated (\( S_2 \)) as long as the timeout expires or one of two negotiating parties withdraws. Re-negotiating an SLA could modify its corresponding running service (\( S_4 \)). Furthermore, when an SLA is completely fulfilled, it will be automatically terminated (\( S_6 \)).

Sometimes, a potential SLA violation could be avoided in execution state (\( S_5 \)). Adding extra resources on the running service, indicating the provider’s dynamic policy with regard to the additional measures to take, is able to safeguard the respective guaranteed quality of a certain specific SLA so that a violation (\( S_7 \)) could be resolved in execution state (\( S_8 \)). Thereby, based on the state machine in Figure 4, an autonomous SLA violation-filtering framework is presented. In Figure 5, three modules are classified and defined individually into three layers.

Specifically, a common SLA manager module creates and maintains one or more SLA modules. Similarly, each SLA module creates and maintains one or more QoS property modules. In this framework, each module has exactly one supervisor, which is the module that creates it. If one module does not have the means for dealing with a certain situation, it will send a corresponding event-driven failure message (e.g., exception) to its supervisor, asking for help. The recursive structure then allows handling failure at the right level. Everything in this framework will be designed to work in a complete distributed environment. As thus, all interactions of modules are pure message passing and asynchronous. Therefore, the SLA violation could be efficiently eliminated.
Consequently, each module will strive to resolve the (potential) violation on its own layer. At very beginning, when a certain violation (e.g., service performance issue) is detected by property module, it tries to fix the violation by restarting the service or adding extra resources on the running service and without altering the content of SLA. In this case, customer does not feel any change and the running SLA is still valid.

Otherwise, it will send an exception message to its supervisor (SLA module). In SLA module, the corresponding exception handlers have already been predefined with all possible solutions. Likewise, if SLA module cannot find an alternative solution, it will forward the message to its supervisor (SLA manager module). SLA manager tries to initiate a renegotiation with the customer so as to establish a new SLA without paying the penalty ($S_5$). And there could also be several rounds of renegotiation within a certain time span.

6. SLA-BASED RESOURCE ALLOCATION AND PROVISIONING

Avoiding SLA violation by initializing an SLA renegotiation is mainly dependent on customer’s willing. In this paper, we are more focusing on how to fix the violation on the first two layers. Through VM live migration, both VM consolidation and service performance can be coordinated. Nevertheless, short downtimes of service migration are unavoidable due to the overheads of moving the VMs. Hence, the respective service interruptions in IaaS reduce the overall service availability and this could also be the main cause of the chain effect on penalties between domains. Here, the term downtime is used to refer to periods when a service is unavailable and fails to provide or perform its primary function to customers. The downtime can be further classified to be planned and unplanned. Since unplanned downtime, e.g., failure of the system, is complicated and uncertain in a simulation environment, in our work, we only consider the planned downtime for evaluating the service availability. The downtime that is introduced by VM live migration is a kind of planned downtime. Thus, the availability is formulated as below:

\[
\text{availability} = \frac{T_a}{T_a + T_b} \times 100
\]  

where $T_a$ is service uptime and $T_b$ is service downtime.

When a user is consuming a service, the service provider has to ensure that the service availability is in accordance with the one expressed in the SLA, in order to avoid the potential penalties due to SLA violations. In this paper, we are focusing on how to manage the number of live migrations so as to control availability according to the established SLA during resource allocation. The optimization of resource allocation problem in a data center can be executed in two steps: initial selection of VMs that need to be migrated, then the chosen VMs will be placed on the hosts using a VM allocation algorithm.

6.1 VM Selection
In Algorithm 1, the input value is the requested availability of customer and the output value is final selected VM that will be migrated. Firstly, all the migratable VMs will be selected by removing the VMs that are already in migration. Thus, the selected VMs will be sorted in descending order of their current service availabilities. Then, the availability of each VM in the sorted VMs list will be re-calculated to check whether the availability is still greater than the requested availability or not, when the VM is migrated. Finally, if such a VM can be found, then we will migrate it and update the downtime record of this VM. Otherwise, no VM will be migrated. The complexity of the selection part of algorithm is $O(n)$, $n$ being the number of migratable VMs.

**Algorithm 1 VM Selection-AV**

**Input:** reqAvailability  
**Output:** selectedVM

```plaintext
migratableVms ← getMigratableVms()  
totalTime ← 86400 // 24 hours in seconds  
vmSize ← migratableVms.getSize()  
vm, downtime, totalDowntime, availability ← NULL  
sortByAvailability(migratableVms) // descending order
for v ← 1 to vmSize do
    vm ← migratableVms[i]  
    downtime ← vm.downtimeEstimator()  
    totalDowntime ← vm.totalDowntime  
    availability ← 1 - (totalDowntime + downtime) / totalTime  
    if availability > reqAvailability then
        selectedVM ← vm
        break
    else
        continue
end if
end for
selectedVM.updateDowntime()
return selectedVM
```

6.2 VM ALLOCATION

In Algorithm 2, the inputs are optimal host utilities and selected VMs; the outputs are the hosts to where each selected VM will be migrated. First of all, the overloaded host(s) and the host(s), which is (are) going to be overloaded after allocating the migrated VM, will not be considered. Then, a host, the utility of which is the closest to optimalHostUtility, will be selected. Here, the optimalHostUtility is not a fixed value and will be discussed in Section 7. We want to find the relationship between the utilization of target allocation host and the value of QoS terms.

Specifically, by considering the service availability constraint, our goal is to allocate the VM to a host that provides the least increase of power consumption and service performance violation due to this allocation. The complexity of the allocation part of algorithm is $O(n)$, $n$ is the number of hosts.

**Algorithm 2 VM Allocation-AVL**

**Input:** optimalHostUtility, selectedVM  
**Output:** allocatedHost

```plaintext
minimalDiff ← Double.MAX_VALUE  
hosts ← getHostList(); host, hostUtility, diff ← NULL  
for h ← 1 to hosts.size do
    host ← hosts[i]
    if excludedHosts.contains(host) then
        continue
    end if
    if hostUtility = host.getUtilizationOfCpu()  
    if host.isSuitableForVm(selectedVM) then
        if hostUtility == 0 & overUtilizedAfterAllocated(host, selectedVM) then
            continue
        end if
        if diff < minimalDiff then
            minimalDiff ← diff
            allocatedHost ← host
        end if
    end if
end for
return allocatedHost
```

6.3 VM LIVE MIGRATION DOWNTIME ESTIMATOR

As discussed in Section 2, the overall duration and short downtime that are introduced by VM live migration are essential properties while implementing service availability in an SLA. In this section, we introduce a VM live migration downtime estimator into CloudSim. As modeled by Akoush et al. (2010) and Salfner et al. (2012), using migration bounds, the downtime of VM live migration is defined in lower and upper bounds as follows:

$$
mig_{\text{overhead}} \leq t_j \leq mig_{\text{overhead}} + \frac{\text{VMsize}}{\text{LinkSpeed}}
$$

(15)

In order to estimate better the downtime value, the authors summarized four main factors that affect the downtime, namely: available link speed, average page dirty rate, VM memory size and migration overheads. The link speed and page dirty rate are proportional to the access traffic of the server applications in a day. The access traffic reaches the highest value at noon and in the morning and late night it will reach the lowest value. Therefore, we assume the probability of determining live migration downtime is consistent with normal distribution as following:

$$
f(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
$$

(16)

where $0 < x < 24$ (hour), expected value is 12 and variance is 1.9, which means at 12 o’clock the server application usually reaches the highest access traffic. For instance, a VM loading server application workloads has 1024 MB memory and 1 Gbps migration link. As such, the lower and
upper bounds of migration are around 314 ms and 9497.8 ms respectively.

7. CASE STUDIES AND DATA ANALYSIS

We choose CloudSim as our simulation platform in order to validate the approaches in Sections 6.1 and 6.2. As it was explained by Beloglazov et al. (2012), CloudSim is able to model and trace the energy consumption and SLA performance with automatic host over / under-loading detection, which reduce the preparation of our simulation work mainly to focusing on SLA availability based VM selection and allocation strategies.

Based on the cloud infrastructure and workloads at GWDG in Germany, a virtual data center is simulated, including 120 virtual hosts. 81 VMs (2 Euro for each) are created, in which 244 containers (3 containers for each VM)
are generated with the corresponding 732 application workloads (3 workloads for each container). 1 kw/h of electricity costs 0.2 Euro. Each container and application workload will make a 2-Euro profit respectively. The whole simulation time is 24 hours. Once the cloud environment in CloudSim is setup, it will automatically allocate the workloads into the VMs. The interval of utilization measurements is 5 minutes. As it was already defined in CloudSim, SLA violation Time per Active Host (SLATAH) is the percentage of time, during which active hosts have experienced the CPU utilization of 100%. And Performance Degradation due to Migrations (PDM) is the overall performance degradation by VMs due to migrations. SLA performance violation, modeled by Beloglazov and Rajkumar (2011) is as following:

\[ SLAV = SLATAH \cdot PDM \]  

(17)

By applying the Algorithm 1 into the CloudSim, we want to test how the SLA availability constraint affects energy consumption and SLA performance. Therefore, the difference between 99.90% and 99.99% is equally divided into 40 intervals, and we set each interval as the reqAvailability of Algorithm 1. The results are illustrated in Figure 6 (a-e). In Figure 6 (a), when we set SLA availability constraint as 100%, meaning VM live migration is not applied, 88 of 120 hosts are always in active, thus all VMs have sufficient resources for their applications without SLA performance violation. However, this leads to huge energy consumption (around 85 kw/h, see upper-right corner of the figure). Once VM live migration is applied, VM consolidation is always considered in order to save on energy consumption. Thus, for example, at 99.99% constraint, 50 hosts are turned into energy saving mode and the energy consumption decreases dramatically (around 52 kw/h). On the contrary, when the availability constraint is not strict (e.g., 99.90%), the SLATAH value (Figure 6 (c)) is relatively low, because as long as an over-load situation is detected, it will be resolved by migrating the VM(s) to the other host(s). Nevertheless, the corresponding PDM value (Figure 6 (b)) is very high, because:

- In under-loaded situations, if all the VMs on this host can be migrated to other host(s), the number of VM migrations is increased. However, this leads to “circular flow” for some VMs, meaning they are migrated between hosts back and forth. Hence, although servers shutting down count increases, energy is actually not saved.
- Extra migrations will definitely lead to performance degradation, therefore the PDM value is also increased.

By applying Equation 17, the final SLA performance violation is as illustrated in Figure 6 (d). From Figure 6 (e), it is apparent that the server shutdown time decreases with increases in service availability. Therefore, from the first experiment, we find that VM live migration can efficiently reduce the energy cost of data center. However, the number of migrations should be balanced in order to achieve the desired service availability and performance requirements.

By applying Algorithm 2 into CloudSim, we strive to find which host will be the optimal destination for allocating the migrated VM(s). Similarly, the host utility between 0% and 100% is divided equally into 200 intervals, and we set each interval value as the optimalHostUtility value into Algorithm 2. By default, 80% is the threshold utility. As illustrated in Figure 6 (f-g), migrating a VM to a host whose utility is the closest to the threshold utility, will lead to the least energy consumption and SLA performance violation. Because otherwise, some VMs from the source host will be migrated back and forth until one of them cannot be moved anymore. This not only increases the number of VM migrations (see Figure 6 (h)) unnecessarily and blocks reasonable future migrations due to the availability constraint, but also introduces further energy consumption.

From Figure 6 (i), we can see that the server shutdown time also decreases with increases in target host utilization. As such, we can directly replace the optimalHostUtility with thresholdHostUtility in Algorithm 2.

In our final experiment, by taking the results of the experiments above, we set 99.99% as target service availability and 80% as target host CPU threshold utilization during VM live migration. Using our workloads, we compare the VM allocation algorithms (THR, IQR, MAD, LR and LRR) and the VM selection algorithms (MMT, RS and MC) of CloudSim with our approach, namely AVL/AV. Consequently, the results (see Figure 7 (a)) show that on average the current resource allocation algorithms in CloudSim are able to deliver service availability for each VM from around 99.7% to 99.93%. In our approach, the service availability is always kept as 99.99%, the SLA violation is 0.00155% and the energy consumption is 54.22 kw/h. Whereas, the other approaches in CloudSim introduce around 37 to 51 kw/h energy consumption and higher SLA performance violation (Figure 7 (b)). Although AVL/AV introduces slightly more energy than the other approaches, using VM consolidation in general still saves much more energy (85 kw/h in non-consolidation case).

The penalty model in Section 4 applies when SLA violation happen. The VM allocation algorithms, using THR, IQR or MAD as VM selection strategy, will lead to availability lower than 99.90%. In this case, the penalty of all three approaches will be increased up to the full cost. Thereby, we selected all the representative approaches to compare with ours in order to find the penalty chain influences in three cloud domains. In Figure 7 (c), our approach introduces the fewest penalties. Whereas by using other approaches, penalties are increased exponentially in PaaS and SaaS layers. Especially, the THR approach will return the full cost of the service due to the availability.
violation-filtering framework. Since Akka is such a framework with features such as message passing, scalability, fault-tolerance and high performance as agreed upon with the customer, the SLA violation can be avoided maximally by representing and monitoring the SLAs in a fault-tolerant fashion. Thus, the results of simulation can be used to prove the correctness and accuracy of emulation.

In order to set performance monitoring policies in an SLA to alert the SLAM when a server nears the threshold of satisfactory performance as agreed upon with the customer, a light-weight hierarchical system will be chosen to represent and monitor the SLAs in a fault-tolerant fashion. As such, the SLA violation can be avoided maximally by resolving the problem in the right component. Since Akka (2013) system is such a framework with features such as message passing, scalability, fault-tolerance and high availability, we plan to apply it into our autonomous SLA violation-filtering framework.

8. CONCLUSIONS

In this paper, we present an OpenStack version of the Generic SLA Manager, which could be further applied into project PaaSage. Based on the proposed autonomous SLA violation-filtering framework, by combining IaaS (OpenStack-SLAM) and PaaS (OpenShift) as a use case, applying a multi-domain SLA pricing & penalty model and introducing a resource allocation strategy, we show experimentally that we can manage VM live migration more efficiently than current state of the art.

In the future, based on the above simulation results, we would like to take this work one step further by using the cloud emulation tool “Emusim” to automatically extract information from various types of workloads (e.g., CPU intensive, memory intensive, network intensive etc.) via emulation and then uses this information to generate the corresponding simulation model. Thus, the results of emulation can be used to prove the correctness and accuracy of simulation.

In order to set performance monitoring policies in an SLA to alert the SLAM when a server nears the threshold of satisfactory performance as agreed upon with the customer, a light-weight hierarchical system will be chosen to represent and monitor the SLAs in a fault-tolerant fashion. As such, the SLA violation can be avoided maximally by resolving the problem in the right component. Since Akka (2013) system is such a framework with features such as message passing, scalability, fault-tolerance and high availability, we plan to apply it into our autonomous SLA violation-filtering framework.

9. ACKNOWLEDGMENT

The research leading to these results is supported by Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) in Germany.

10. REFERENCES


Kuan Lu is working as a scientific researcher and PhD candidate of the GWDG at the University of Göttingen, Germany. He received his Master degree in computer engineering from Duisburg-Essen University, Germany. His expertise lies within planning and optimization for service level agreements of cloud computing, resource scheduling and allocation management, parallel computing, service computing. He has been actively involved in the FP7 IP PaaSage, SLA@SOI projects and the projects of Lotus Mashup and MILA at IBM Research and Development GmbH, Böblingen, Germany.

Prof. Ramin Yahyapour is the managing director of the GWDG at the University of Göttingen, Germany. He is active in research on clouds, grid and service oriented infrastructures for several years. His research interest lies in resource management. He is a steering group member and on the Board of Directors in the Open Grid Forum. He has been active in several national and European research projects. Notably, he was the scientific coordinator of the FP7 IP SLA@SOI and was a steering group member in the CoreGRID Network of Excellence.

Constantinos (Costas) Kotsokalis received his electrical and computer engineering diploma from the National Technical University of Athens, Greece, and his PhD from the Dortmund University of Technology, Germany. His expertise lies within the domain of distributed computing at large, with a focus mostly on service level agreements. Additional scientific topics of interest include distributed algorithms, job / resource scheduling and data management. Costas has worked in multiple international projects and diverse domains such as wide-area networking, grid / cloud computing, SLA management, remote instrumentation and e-Health.

Philipp Wieder is deputy leader of data center of the GWDG, Germany. He received his PhD from the Dortmund University of Technology, Germany. He has been active in research on clouds, grid and service oriented infrastructures for several years. His research interest lies in distributed system, service level agreements and resource scheduling. He has been actively involved in the FP7 IP PaaSage, SLA@SOI and SLA4D-Grid projects.

Edwin Yaqub is working as a Research Assistant and PhD candidate of the GWDG at the University of Göttingen, Germany. He received his Master degree in computer science from RWTH Aachen Technical University, Germany. His research expertise includes SLA Negotiations for cloud and service computing. He has been actively involved in the FP7 IP PaaSage, SLA@SOI projects. Previously, he worked as a software engineer on e-Commerce, web and desktop based business applications. During studies, he worked part time at the University, Philips Research Labs and Siemens Research, Aachen, Germany.

Ali Imran Jehangiri studied in Bergische Universität Wuppertal, where he did Master in computer simulation in the disciplinary science of experimental particle physics. He later joined the chair of service computing at Dortmund University of Technology, Germany. Currently, he is a PhD candidate of the GWDG at the University of Göttingen, Germany. His research interest is in monitoring of service level agreements for virtual platforms.