University of Zurich[UZH]

Patrick Poullie, Stephan Mannhart, and Burkhard Stiller

Defining and Enforcing Fairness Among Cloud Users by Adapting Virtual Machine Priorities During Runtime

TECHNICAL REPORT – No. IFI-2016.04

March 2016

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Virtual Machine Priority Adaption to Enforce Fairness Among Cloud Users

Patrick Poullie, Stephan Mannhart, Burkhard Stiller
Communication Systems Group (CSG), Department of Informatics (IfI), University of Zürich (UZH),
Binzmühlestrasse 14, CH–8050 Zürich, Switzerland
Email: poullie@ifi.uzh.ch, stephan.mannhart@uzh.ch, stiller@ifi.uzh.ch

***Abstract*—In recent years fairness problems in data centers have been pointed out and job/Virtual Machine (VM) scheduling has been chosen as a solution approach. Clouds are a special case of data centers, where resources are deployed by VMs in a highly dynamic manner during VM runtime. However, scheduling only allows influencing resource allocations, when VMs are instantiated, *i.e.*, before runtime. Thus, runtime prioritization bears a great potential to manage cloud resources and promote fairness in clouds, especially, when VMs run over long periods. Nevertheless, runtime prioritization is not leveraged accordingly.**

This paper defines fairness as handicapping VMs of heavy users during runtime to allocate more resources to VMs of light users. Thereby, the need to make assumptions on user's utility functions is avoided, while different fairness notions can be captured by adapting the definition of heaviness. Guidelines for this definition are provided to ensure incentives to configure and utilize VMs adequately. Finally, OpenStack is extended in its implementation by a decentralized fairness service to enforce fairness according to this definition. The fairness service's functionality is certified by experiments in terms of overhead and fairness promotion.

I. Introduction

Cloud computing allows server farms to provide their combined computing power on demand to numerous users. These users start VMs that are hosted by the cloud's nodes. Since cloud nodes are shared by VMs of different users, Physical Resources (PR), such as CPU time, RAM, disk I/O, and network access, become subject to conflicting interests [7]. While in public/commercial clouds resource allocation to VMs is prescribed by Service Level Agreements (SLA) [5], private clouds, clusters, and grids are often a commodity. Thus, it is important to ensure fairness, when allocating resources of these commodity infrastructures, especially in those cloud computing environments, where financial incentives for users do not necessarily exist directly [24].

Cloud computing resource allocation consists of the following two steps, which are conducted continuously and in parallel:

VM scheduling: During this step the cloud's orchestration layer decides which VM is started next and which node hosts the VM. This step is conducted for every VM that is started or live migrated.

Runtime Prioritization: During this step a node's hypervisor, *i.e.*, the node's operating system, allocates the node's PRs, to the VMs. This step is conducted permanently.

Thus, the first step decides, which node hosts a VM, and the second step decides how many PRs this node allocates to this VM. In particular, because CPU time, disk I/O, and network access are time-shared, the second step allows to allocate efficiently these PRs by Proportional Priorities (PP). Only for RAM, which is space-shared, the (re)allocation may come with a certain overhead. However, this (re)allocation still outperforms live-migrating of a VM, as live migration implies transferring the entire node's RAM content to another node. Thus, the second step allows to flexibly manage cloud resources by changing the priorities of running VMs. For example, a VM scheduled to a weak node may perform better than when scheduled to a powerful node, if the weak node prioritizes this VM.

Accordingly, both allocation steps are important for the cloud resource allocation process. Although runtime prioritization bears great potential to manage the resource allocation in clouds, VM scheduling is leveraged significantly more often. This is problematic, since scheduling only allows changing the order in which VMs are instantiated and is, therefore, inefficient to manage resource allocation among users that run VMs over long periods. Also when enforcing fairness, which is particularly relevant in private clouds, this imbalance of leveraging the two steps applies. Two key reasons for this imbalance are that (i) fair runtime prioritization is harder to define than fair VM scheduling and (ii) leveraging runtime prioritization to enforce fairness gives rise to several implementation details, such as monitoring how many resources VMs utilize, aggregating this information to users, and determining tools to set VM priorities.

This paper addresses both problems. To tackle (i), fairness is defined as "handicapping VMs of heavy users during runtime to allocate more PRs to VMs of light users." This definition avoids making assumptions on users' utility functions, as utility functions during VM runtime are highly fluctuant and often unavailable due to technical or privacy constraints. By adapting the definition of heaviness different fairness notions can be captured and incentives provided to users to configure and utilize their VMs appropriately. To tackle (ii), OpenStack's implementation is extended by a fairness service that enforces fairness according to this definition.

Fairness is particularly important in private clouds. However, being able to prioritize VMs in a fair manner during VM runtime also allows commercial clouds to introduce much

TABLE I: Abbreviations frequently used in this paper

CLI	Command Line Interface
CRS	Cloud Resource Supply ($\sum_{h_j \in N} r(h_j)$)
BBF	Bottleneck-based Fairness
DRF	Dominant Resource Fairness
FS	Fairness Service
NRI	Node Resource Information ($r(n)$ for $n \in N$)
PR	Physical Resource
PP	Proportional Prioritiy
RUI	Resource Utilization Information ($l(v)$ for $v \in V$)
SLA	Service Level Agreement
VM	Virtual Machine
VR	Virtual Resource

simpler charging schemes for cloud services, such as cloud flat rates, where users pay a fixed price for a certain quota from which they can instantiate VMs [24].

Clouds are a special case of data centers, which also comprise clusters and grids. Dominant Resource Fairness (DRF) and Bottleneck-based Fairness (BBF) are the most prominent approaches to multi-resource fairness in data centers and address the problem of fair VM scheduling. Both make the simplifying assumption that resources are required in static ratios and define how a fair allocation looks like under this assumption. However, while even during VM scheduling ratios in which resources are required may change, this is even more likely, during runtime priotirization. Thus, this work here distinguishes itself from DRF and BBF, as it (i) addresses the problem of fair runtime prioritization (and not VM scheduling), (ii) is, therefore, complementary to works on DRF and BBF, and (iii) does not make assumptions about utility functions. Furthermore, fairness is not defined as a concrete allocation but as the procedure of prioritizing light users at the cost of heavy users. Lastly, while DRF and BBF only consider one consumption vector per user, the approach presented here also takes user quotas and how users have configured their VMs (in order to give incentive to configure VMs properly) into account.

The remainder of this paper is structured as follows: Section II discusses related work and proves that the approach to cloud fairness taken here is novel. Section III describes the organization of cloud resources to conclude on an intuitive definition of fairness being applicable during runtime, which leads to the cloud user heaviness definition. While Section IV describes the implementation of the OpenStack fairness service to enforce the new fairness notion, the implementation is evaluated in Section V. Finally, Section VI concludes the paper and outlines future work. Table I lists abbreviations used throughout the paper.

II. Related Work

Multi-resource Fairness issues in virtualization environments were first pointed out by [10]. As a solution, [10] defines Bottleneck-based Fairness (BBF). [8], [14] provide theoretical insights on this concept and [31] details how BBF can be implemented between processes that share CPU time, network access, and disk I/O. The most prominent definition of data center fairness is Dominant Resource Fairness (DRF) as introduced in [12]. DRF has been extended in many directions, for example by indivisibilities of resources and user hierarchies [22], [11], [32], [2]. A third approach to data centers fairness is the extension of Proportional Fairness [9] to multiple resources [3], [4]. All three approaches (BBF, DRF, proportional fairness) target scheduling and assume that resources are required in static ratios. Thus, these approaches cannot be applied to runtime prioritization of VMs, as here resource dependencies are unpredictable. Only [31] describes how to theoretically achieve BBF among running processes with varying demands. However, it is also unsuited to be applied in clouds, as it solves the problem by fine-granular resource scheduling.

Because resources during runtime have to be allocated by assigning priorities to VMs, functions that map (multi-resource) consumption vectors to (priority) scalars are better suited. This also avoids the need for assuming utility functions. [16], [15] present such priority functions and apply these functions to scheduling. Although these functions are generally applicable to runtime prioritization, they do not allow to take the Virtual Resources (VR) of VMs into account, as they only operate on consumption vectors but not VR vectors. However, taking VRs into account is important to give incentive to users to configure their VMs correctly and, thereby, allow for most efficient resource utilizations.

[17], [18] is closest to this work here. Users can trade resources with other users and adapt their VM runtime priorities. The adopted fairness notion is asset fairness. [17], [18] requires trading mechanisms and VM demand prediction.

Thus, those existing approaches either (i) make assumptions on utility functions that are unrealistic during runtime, (ii) prohibit giving incentives to users to configure VMs correctly, or (iii) require multiple complex mechanisms. These three aspects are addressed in the novel theoretical approach proven applicable by the evaluation of an OpenStack-based implementation.

III. Problem and Approach

A cloud consists of a set of users $U = \{u_1, u_2, \ldots, u_x\}$, a set of nodes $N = \{n_1, n_2, \ldots, n_y\}$, and a set of VMs $V = \{v_1, v_2, \ldots, v_z\}$. VMs are started by the users to process varying workloads. Each VM is hosted by a node, whereat the cloud scheduling policy (and not the user) decides which node hosts a VM. Function $o\colon V \to U$ maps a VM to its owner, *i.e.*, the user that started the VM, and $a\colon V \to N$ maps a VM to the host that accommodates/hosts the VM.

VMs share heterogenous PRs such as CPU time, RAM, disk I/O, and network access. Let $R = \{r_1, r_2, \ldots, r_m\}$ be the set of PRs to be considered for a fair allocation. VMs are defined by Virtual Resources (VRs), *e.g.*, virtual CPU (VCPU) and virtual RAM (VRAM), often chosen from a range of different *flavors*, *i.e.*, a VM flavor is a set of VRs that a VM of that flavor has. Especially in private clouds, resources may be managed by quotas, *i.e.*, each user has a quota that defines a maximum of VRs that the user's VMs may have in total. Function $r\colon V \cup N \cup U \to \mathbb{R}^m_{\geq 0}$ maps (i) VMs to their VRs,

(ii) nodes to their PRs, and (iii) users to their quota. Function $l: V \rightarrow \mathbb{R}^m_{\geq 0}$ maps VMs to the load they impose on the PRs (at a distinct point in time). Arithmetic operations on vectors are applied point-wise.

A. Problem Statement

Cloud runtime fairness cannot be uniquely defined [24]. The reason is that heterogenous PRs are shared, while users have different demands. For example, some users may require more CPU for their workloads while others require more RAM. A third user may deploy the cloud for backups, which requires mostly disk-space and bandwidth. Thus, resource shares are not objectively comparable.

In economics, the problem of defining fairness is solved by definitions based on consumer's utility functions (a consumer's *utility function* maps each bundle to a number quantifying the consumer's valuation for the bundle). However, because cloud user demands (and, therefore, utility functions) change frequently, such definitions can neither be applied nor enforced.

Also, the utilization of PRs is a continuous process and the allocation of most PRs is managed by weights/shares, which are referred to as Proportional Priorities (PP) in this paper. Therefore, the amount a VM receives of a PR cannot not be configured but instead PPs have to be assigned such that the designated allocation is approximated.

B. Runtime Prioritization Approach

Given the above constraints, this paper defines cloud fairness as the procedure of *prioritizing cloud users inversely to their heaviness, whereat this heaviness is determined by the stress the user imposes on the cloud.* As VMs and thereby users utilize different heterogenous resources, this definition requires a cloud user heaviness metric $h_u: U \rightarrow \mathbb{R}$ that quantifies the stress the user imposes on the cloud by summing up the heaviness of the user's VMs. The heaviness of a VM is quantified by VM heaviness metric $h_v: V \rightarrow \mathbb{R}$. Metric h_v can be defined by a function $d: \mathbb{R}^m_{\geq 0} \rightarrow \mathbb{R}$ that maps the VM's multi-resource load vector to a heaviness scalar, as for example the functions presented in [15], [16] or functions that define the value of a vector as asset fairness or DRF do (see [**?**] for how to define h_u such that DRF is captured). However, such definitions are insufficient to (i) adequately represent the stress the VM imposes on the cloud and (ii) provide incentive to users to configure and utilize VMs appropriately.

1) Rewarding Correct VM Configuration: A VM's VRs are an indicator of which PRs the VM will utilize during runtime. Accordingly, the cloud budgets a certain amount of PRs for the VM based on the VM's VRs. Therefore, a VM's VRs determine which node is selected as the VM's host. For example, placing "small" VMs on nodes with less remaining capacity increases the utilization of these nodes and leaves nodes with more remaining capacity free to accommodate "large" VMs that cannot be hosted by nodes with less remaining capacity.

As VMs are scheduled based on their VRs to optimize node utilization, a VM with a load that strongly deviates from what is anticipated based on the VM's VRs, leads to either over-loaded or under-utilized PRs on the VM's host, *i.e.*, higher stress for the cloud. Accordingly, h_u must give incentive to users to choose the VRs of their VMs properly, *i.e.*, the heaviness of a user must decrease the more the user's VMs' VRs conform with the VMs' load. This is referred to as the *configuration incentive* subsequently.

2) Discouraging Idle VMs: Let two users $u_a, u_b \in U$ utilize the same amount of PRs, *i.e.*, the sum of the load their VMs produce is equal. User u_a produces this load by one busy VM, which utilizes 50 times the PR amount compared to its idle state. In contrast, u_b has instantiated 50 VMs of the same flavor, which are idle. In order to be able to provide for u_b's VMs if necessary, the cloud budgets a large amount of PRs. Although over-provisioning makes this factor of budgeted PRs smaller than 50, u_b stresses the cloud notably more than u_a.

This example can be viewed as a special case of the problem outlined in Section III-B1 in the sense that v_b has 50 VMs that are poorly utilized compared their VRs. However, it additionally reveals that, while h_v must take the VRs into account to calculate the "penalty" for the load a VM produces (cf. Section III-B1), it must also statically account for the VRs irrespective of the VM's load. This gives incentive to users to avoid operating idle VMs and is therefore referred to as the *utilization incentive* subsequently.

C. Heaviness

To provide the incentives as discussed above, the heaviness metric h_u is defined as follows.

1) Required Resource Information: The most important factor that determines the heaviness of users are the PRs users' VMs utilize. Therefore, resources utilized by every VM during runtime must be collected. This information (which is denoted by the function l) is referred to as a VM's Runtime Utilization Information (RUI). To provide the configuration and utilization incentive, all VMs' VRs are required as input.

Resources are heterogenous and measurable in different units. Thus, a cloud-wide unique normalization is necessary to map a multi-resource consumption vector to a heaviness scalar. This normalization is done by the Cloud Resource Supply (CRS) $:= \sum_{n_j \in N} r(n_j)$. To calculate the CRS and, thereby, the normalization vector, the PRs of all nodes are required, which is referred to as Node Resource Information (NRI).

In summary, the heaviness metric requires each node's NRI as well as each VM's (i) RUI, (ii) VRs, (iii) host, and (iv) owner as input.

2) Scales and Overcommitment: Overcommitting resources allows clouds to achieve high utilization. Overcommit ratios determine the factor by which the sum of VRs of a node's VMs can exceed the node's PRs. For example, when the cloud's CPU overcommit ratio is 15, a node with 4 CPU cores can host VMs with at most 60 VCPUs in total. Higher overcommit ratios increase the degree of cloud utilization and the chance of overload. Due to this utilization-overload-tradeoff, there is no best overcommit ratio. Thus, overcommit ratios differ not only from cloud to cloud but also from resource to resource. Due to overcommitment, the sum of quotas is allowed to exceed

the CRS and the sum of VRs of VMs hosted by a node is allowed to exceed the node's PRs. In other words, quotas and VRs are on the *virtual scale* and PRs are on the *actual scale*.

3) VM Endowments: The configuration incentive demands that h_u quantifies the heaviness of a user lower, when the user configures VMs in conformance with the subsequent workload. The rationale behind this is that based on a VM's VRs the cloud expects a certain load of the VM and schedules the VM on a node that can most economically provide for this anticipated load. In particular, depending on the cloud's overcommit ratios and a VM's VRs, the cloud budgets a certain amount of PRs for the VM. Therefore, the better a VM's load conforms with budgeted PRs, the less it must increase the heaviness of its owner. Accordingly, a VM's *endowment* is defined as the amount of PRs that are budgeted for this VM. This definition allows for multiple functions $e\colon V \to \mathbb{R}^m_{\geq 0}$ to calculate the endowment. Subsequently, three reasonable functions are discussed.

The straight-forward endowment function is to scale a VM's VRs by the ratio of the CRS to the number VRs that can maximally be assigned to VMs, *i.e.*,

$$e(v_i) \mapsto \frac{\sum_{n_j \in N} r(n_j)}{\sum_{u_k \in U} r(u_k)} \cdot r(v_i).$$

However, this formula does not account for the number of VMs that are actually instantiated. This can be overcome by defining the endowment function to scale a VM's VRs by the ratio of the CRS to the VRs of running VMs, *i.e.*,

$$e(v_i) \mapsto \frac{\sum_{n_j \in N} r(n_j)}{\sum_{v_k \in V} r(v_k)} \cdot r(v_i).$$

However, depending on the ratio of VR the VM will be scheduled to different nodes, wherefore the VM's host's PRs are the best indicator of how many resources are budgeted for a VM. Accordingly, this paper recommends defining the endowment of a VM as the VM's proportional share of the host's PR:

$$e(v_i) \mapsto \frac{r(a(v_i))}{\sum_{v_j \in V : a(v_j) = a(v_i)} r(v_j)} \cdot r(v_i). \tag{1}$$

While VRs are on the virtual scale, all three endowment definitions map VRs to the actual scale.

4) VM Heaviness: The heaviness of a VM v_i is defined by

$$h_v(v_i) := s(e(v_i)) + d(v_i). \tag{2}$$

Function $s\colon \mathbb{R}^m_{\geq 0} \to \mathbb{R}_{\geq 0}$ defines the *static heaviness* to introduce a cost for instantiating a VM irrespective of its load only based on its VRs. Function s receives the VM's endowment as input, which can be viewed as VRs mapped to the actual scale (cf. Equation 1), and provides the utilization incentive. To account for the size of a VM, s must strictly increase with each input dimension.

Function $d\colon V \to \mathbb{R}$ defines the *dynamic heaviness* of a VM and represents the cost of providing for the VM's load. Let $x, y \in \mathbb{R}^m_{\geq 0}$ and $v_i, v_j \in V$ with $l(v_i) = l(v_j) = x + y$, $e(v_i) = x$ and, $e(v_j) = x + y$, *i.e.*, VMs v_i and v_j utilize the same amount of PRs and v_j's endowment conforms perfectly with that utilization, while v_i's endowment is too small. Due to v_i's and v_j's endowments $s(v_j) > s(v_i)$ holds true. It is crucial to choose functions s and d, such that $h_v(v_i) > h_v(v_j)$, which is referred to as the *non-minimalistic condition*. If the non-minimalistic condition would not hold, the static heaviness would outweigh the dynamic heaviness of a VM, wherefore the heaviness of a VM could be minimized by minimizing the VM's VRs irrespective of the VM's load. This would violate the configuration incentive.

It is important that the heaviness of a VM v_i is not less than zero, as it would give incentive to users to instantiate more of these VMs. As the static heaviness is always greater zero, it is feasible that the dynamic heaviness is less or equal to zero as long as $d(v_i) > -s(e(v_i))$. In particular, one may wish to define the dynamic heaviness such that it is zero, when $l(v_i) = e(v_i)$, as this indicates the equilibrium between VM utilization and what is budgeted for the VM. In this case, it is important that the dynamic heaviness is less than zero, when the VM utilizes less than its endowment. Otherwise, users would have no incentive to reduce the utilization of their VMs below the VMs' endowment.

By choosing d and s, such that $d(v_i) > -s(e(v_i))$ and the non-minimalistic condition hold, the configuration and utilization incentive are provided.

5) User Heaviness: Data center users can be heterogenous. In particular, depending on the payment of users or other differentiation criteria, users can have different quotas. Let two users $u_1, u_2 \in U$ instantiate an identical VM. Let u_1 have a larger quota than u_2. Then, the heaviness of u_2 must be larger, since u_2 utilizes the same resources as u_1 but has a smaller entitlement to the cloud's resources. Therefore, a user's heaviness must decrease with the user's unused quota.

A user's heaviness is largely determined by the heaviness of the user's VMs. The VMs' heaviness is calculated from input on the actual scale (cf. Section III-C4), while the user's quota is on the virtual scale. Thus, the quota has to be mapped to the actual scale, before factoring it into the user heaviness. This practical scaling is done by the *actual quota* of a user u_i, which is defined as the share of the cloud's PRs that is proportional to u_i's quota and calculated by function

$$aq(u_i) \mapsto \frac{\sum_{n_j \in N} r(n_j)}{\sum_{u_k \in U} r(u_k)} \cdot r(u_i). \tag{3}$$

The actual quota is a vector and the heaviness of a user is a scalar. Thus, function $s'\colon \mathbb{R}^m_{\geq 0} \to \mathbb{R}_{\geq 0}$ has to be defined to map the actual quota to a scalar that is subtracted from the user's heaviness. The recommended choice is $s' = s$.

Finally, the heaviness of a user u_i is defined as the sum of u_i's VMs' heaviness subtracted by u_i's unused quota, *i.e.*,

$$h_u(u_i) := \left(\sum_{v_j \in V : o(v_j) = u_i} h_v(v_j) \right) - s'(aq(u_i)). \tag{4}$$

D. DRF and BBF Heaviness

Although BBF and DRF are complementary to the approach taken in this paper, both fairness notions include a metric to quantify what consumers receive. Accordingly, heaviness can be defined according to these metrics.

BBF demands to allocate every user, who is not satisfied, at least what the user is entitled to on at least one bottleneck resource. Let $b \in \{0,1\}^m$ be the vector with $b_i = 1$ for $i \in \{1,2,\ldots,m\}$, if and only if r_i is a bottleneck. Then

$$h_u^{\text{BBF}}(u_i) := \max\left(b \cdot \sum_{v_j \in V : o(v_j) = u_i} l(v_j)/aq(v_j)\right) \quad (5)$$

determines the heaviness of a consumer according to BBF and must be ≥ 0, if u_i is not satisfied.

A user's heaviness according to DRF is determined by the user's dominant share and, therefore,

$$h_u^{\text{DRF}}(u_i) := \max\left(\sum_{v_j \in V : o(v_j) = u_i} l(v_j)/\text{CRS}\right). \quad (6)$$

h_u^{DRF} and h_u^{BBF} cannot be captured by h_u, as both definitions apply a function to the total consumption of a user, while h_u maps each VM consumption to a scalar and adds up these scalars. The reason that h_u is unable to capture h_u^{DRF} and h_u^{BBF}, is that h_u^{DRF} and h_u^{BBF} are designed for a different purpose. In particular, h_u^{DRF} and h_u^{BBF} are designed for VM scheduling and not runtime prioritization and, therefore, are unable to factor in the configuration of VMs or the user quotas. Accordingly, h_u^{DRF} and h_u^{BBF} have the downside of not being able to provide the configuration or utilization incentive that h_u is able to provide.

E. Decentralization and Complexity

This section discuses how heaviness according to the presented definition is calculated in a decentralized manner. The resulting message volume, *i.e.*, the sum of the size of exchanged messages, is calculated. It is assumed that each node applies function h_v to calculate the heaviness of the hosted VMs, announces this *heaviness set* to all other nodes and accordingly receives the heaviness sets of other nodes. Each node then applies function h_u to the calculated and received VM heaviness sets to calculate the heaviness of users.

The CRS is essential to calculate the heaviness of VMs and users (cf. Section III-C1). As the CRS is the sum of nodes' PRs, every node needs to broadcast its PRs, such that every node can calculate the CRS. It is sufficient, to perform this broadcast once, as the PRs of a node are static. In case a node is added to the cloud, it announces its PRs to every node, to which every node responds with its PRs. Accordingly, each node once has to send a vector $r \in \mathbb{R}_{\geq 0}^{|R|}$ to every other node. This results in $|N|\cdot(|N|-1)$ messages with a size of $|R|$ being send. The *CRS message volume*, therefore, is $|N|\cdot(|N|-1) \cdot |R| \in \mathcal{O}(|N|^2 \cdot |R|)$.

When a node knows the CRS it can locally calculate the heaviness of each hosted VM. The resulting heaviness set has to be broadcasted to the cloud, such that every node can calculate the heaviness of each user. This results in $|N|\cdot(|N|-1)$ messages (assuming every node hosts at least one VM) being send. In particular, each node n_i has to send a message m to the $|N|-1$ other nodes. Message m contains $|\{v \in V | a(v) = n_i\}|$ scalars, wherefore, the size of m is only bounded by $|V|$. Nonetheless, as the heaviness of every VM is, thus, send to $|N|-1$ nodes, the *heaviness message volume* is $(|N|-1) \cdot |V| \in \mathcal{O}(|N| \cdot |V|)$.

In order to calculate the heaviness of any user u_i, nodes also need to know $s'(aq(u_i))$, which is a static scalar. Accordingly, $s'(aq(u_i))$ must be announced initially to every node by the central entity that holds this information. Thus, the *quota message volume* is $|N| \cdot |U| \in \mathcal{O}(|N| \cdot |U|)$.

The CRS and quota message volume are exchanged once and it is reasonable to assume that $|R|$ is small and constant and that $|V| \gg |N|$ and $|V| \gg |U|$, because users usually own and nodes usually host several VMs. Accordingly, the message volume that needs to be exchanged to keep the heaviness of users up to date is $\mathcal{O}(|N| \cdot |V|)$. Thus, the heaviness message volume that needs to be exchanged consistently to keep the user heaviness up to date is the most significant factor.

All messages send between nodes are broadcasts, *i.e.*, when a message is send to one node, a message with the same content is also designated for all other nodes. Accordingly, the message volume can dramatically be reduced techniques that are used in P2P systemsto make the information exchange and, particularly, broadcasts more efficient. Alternatively, a centralized node, which receives the messages from all nodes, applies the necessary calculations and returns the results to all nodes, reduces the message volume to $\mathcal{O}(|R| \cdot |N| + |V| + |U| \cdot |N|)$.

IV. Implementation

Out of the box, OpenStack does not leverage runtime prioritization to manage resource allocation. This section describes the extension of OpenStack to leverage runtime prioritization in order to enforce fairness among cloud users. Fairness is enforced as discussed in Section III-B and, thus, the extension deploys the definitions presented in Section III-C.

A. High-level Design and Steps

Two essential node types in the OpenStack architecture are the controller node and the compute nodes. The *controller node* coordinates the cloud, *e.g.*, by scheduling VMs, and stores essential information to be accessed by other nodes or administrators. *Compute nodes* perform the actual processing of workloads, *i.e.*, hosting VMs. For this purpose, compute nodes run the OpenStack service `nova-compute` that is part of the OpenStack compute project nova. OpenStack nova knows three types of managing components: Services, managers and drivers [21]. Services are generic containers that group compute node functionalities. Each service has its own managers to control a specific aspect of the node and execute tasks. Managers encapsulate functionalities of the service by providing methods and periodic tasks to deploy the functionality.

Compute nodes have direct access to VMs they host and, therefore, can monitor the RUI of these VMs and adapt VMs' priorities. Thus, runtime prioritization is leveraged by an additional nova service called `nova-fairness`. This Fairness Service (FS) enforces fairness in the cloud as discussed (accordingly, it collects all information discussed in Section III-C1) and allows for the definition of functions d, e, s, and s' in oder to adapt the heaviness definition h_u (cf. Equation 4). The message exchange between FS instances on different nodes is decentralized.

Let $N^f \subseteq N$ be the set of compute nodes that run the FS and $n_i \in N^f$. The pseudocode in Listing 1 describes how the FS running on node n_i calculates the heaviness of users and adapts the priorities of hosted VMs.

```
1  while NRI of some host in N^f is missing:
2      send own NRI to nodes of which NRI is missing
3  use NRIs to calculate CRS and normalization vector
4  every μ seconds:
5      collect RUI of all VMs hosted by n_i
6      apply h_v to collected RUI in order to calculate heaviness
           of all VMs hosted by n_i
7      send this heaviness set to all n ∈ N^f − {n_i}
8      wait to receive heaviness set from all n ∈ N^f − {n_i}
9      apply h_u to calculate the heaviness of all u ∈ U
10     for every VM v hosted by n_i:
11         set priorities of v according to h_v(v) and h_u(a(v))
```

Listing 1: Steps of the FS running on node n_i.

Lines 1 to 3 ensure that the CRS, which is essential to calculate the heaviness, is available before the FS conducts any further steps. In order to allow adding nova-fairness nodes subsequently, the FS responds with its NRI upon receiving the NRI of a new node (this is not reflected in the pseudo code). μ in Line 4 defines the interval with which the heaviness of users and PPs of VMs are updated and is referred to as the *update interval*. Every μ seconds the FS calculates the heaviness of VMs hosted by n_i (Line 5 and 6). This *heaviness set* is announced to all nodes (Line 7). When heaviness sets have been received from all other nodes (Line 8), the node calculates the heaviness of users (Line 9) from this information. Lastly, priorities of VMs on n_i are set according to the calculated heaviness (Line 10 and 11).

B. Resource Measurement

Currently the FS takes the following six resources into account: (i) CPU time in seconds, (ii) memory used in kilobytes, number of bytes (iii) read from disk and (iv) written to disk, and number of bytes (v) received and (vi) transmitted through the network interface.

1) CPU Time Normalization: CPU time, *i.e.*, the amount of time used for a specific CPU task to complete [29], measures CPU usage. CPU time provided by different nodes is not directly comparable. The reason is that cloud nodes are rarely homogeneous [13] and, therefore, are equipped with different CPUs. Accordingly, one second of CPU time on a powerful node is more valuable than one second of CPU time on a less powerfull node. To compare CPU time across nodes, the FS normalizes CPU time by the nodes' BogoMIPS [**?**]. BogoMIPS is a metric provided by the Linux operating system to capture the performance of different CPUs. However, BogoMIPS do not define a scientifically reliable measure to compare CPUs, wherefore other normalization references, such as the SPEC value [28], are considered for future improvements.

2) Resource Utilization Information: The FS requests a list of VMs that are running on its node by an Remote Procedure Call (RPC) to the `nova-conductor` service. This list contains only VMs that have been successfully scheduled by OpenStack on the node. Therefore, the list indicates for which VMs RUI have to be collected (cf. Line 5 of Listing 1). The RUI is collected by deploying an OpenStack driver to access the `libvirt` virtualization API. This API allows monitoring the detailed VM RUI in a unified manner and supports most of the known hypervisors [26]. Accordingly, the libvirt API provides access to the RUI for each VM and ensures the FS's compatibility with numerous hypervisors. For time-shared resources (CPU time, disk I/O, network access), the libvirt API provides the accumulated resource consumption since boot time. Therefore, the FS calculates the RUI for the current update interval by subtracting the accumulated consumption at the beginning of the interval from the accumulated consumption at the end of the interval. For space-shared resources (RAM) the libvirt API provides the current utilization, which the FS uses to represent RAM consumption in the RUI vector.

3) Node Resource Information: The NRI of a node specifies its hardware resource capacities. Therefore, the NRI per node is stable, which allows to capture NRI without a temporal component (Line 1-3 of Listing 1). The different NRI data points considered by the FS are (i) number of CPU cores, normalized by the node's BogoMIPS, (ii) combined disk read speeds of all disks in bytes/s, (iii) network throughput in Byte/s, and (iv) total amount of installed memory in kBytes.

C. Heaviness Metric

Section III-C outlined desirable criteria for a heaviness metric and presented the generic heaviness metric h_u. Metric h_u contains wildcard functions d, e, s, and s', which can be defined within certain constraints. Accordingly, the FS adopts this generic heaviness metric and allows defining these functions. To this end, generic functions are inherited and overwritten by designated definitions. The definitions to be used are specified in the nova configuration by the class path of these definitions. This class path is checked by the FS for correctness, *i.e.*, whether a correct class with required definitions exists under that path.

Flavors in OpenStack do not contain a virtual counterpart for every PR. For example, OpenStack flavors contain VCPU and VRAM but not virtual disk I/O or virtual network access. However, the definitions for function e in Section III-C3 assume that every PR has a virtual counterpart. Therefore, in case a PR has none, the FS divides its node's supply of that PR by the number of hosted VMs. In turn, this result

determines the amount of the virtual counterpart of this PR every VM on that host owns. For example, when a node with a bandwidth of 10 Gbit/s hosts four VMs, each VM's virtual network access is set to 2.5 Gbit/s.

D. Message Exchange

The FS's information exchange between compute nodes (cf. Lines 2, 7, and 8 of Listing 1) is implemented in a decentralized and asynchronous manner.

1) Decentralization: The nova service has a server counterpart on the controller node. In particular, the server hosts a centralized message broker that relays all messages. Therefore, by default, all information exchanged between compute nodes traverses the controller node. This is not scalable and introduces a single point of failure (although, arguably, the controller node is always a single point of failure). To overcome this drawback the message exchange between compute nodes is decentralized, as described subsequently.

OpenStack enables message exchanges between nodes by message queues. OpenStack's default message queuing protocol is `RabbitMQ` [6], which is an implementation of the `AMQP` specification. AMQP is centralized, wherefore using RabbitMQ implies that all inter-node communication traverses the RabbitMQ broker on the controller node that relays messages. To allow for a decentralized information exchange between compute nodes, the `ZeroMQ` [19] message queuing system is deployed. ZeroMQ can be set up with minimal effort and allows compute nodes to exchange information directly by running a decentralized ZeroMQ message broker on each node. Using ZeroMQ results in a message volume of $\mathcal{O}(|N|\cdot|V|)$ that is partitioned to $|N|\cdot(|N|-1)$ messages (cf. Section III-E) every μ seconds.

The only mechanism of ZeroMQ that works centralized is the discovery of nodes to communicate with. For this purpose, a `redis` data structure store [27] hosting identification information for all ZeroMQ brokers has to be installed on the controller node. Because redis is an in-memory data store, it is highly performant and sufficiently scalable.

2) Isochronous Message Exchange: To achieve independence of compute nodes, FS instances on different nodes operate isochronously, *i.e.*, nodes adhere to the same update interval μ, but send messages within this interval at different times. Moreover, messages between nodes are sent by RPC casts, as these, in contrast to calls, do not invoke a response. Accordingly, messages that contain NRI and heaviness sets from other nodes reach a node at different times. To ensure that messages are only sent to nodes, which are currently reachable and running the FS, an according list is requested first from the `nova-conductor` service. The nova-conductor queries a `SQL` database located on the controller, which contains all status information about services currently running.

The NRI is static, wherefore a compute node only needs to receive this information once from every other node. Therefore, the FS waits to conduct any other operations, until the NRI of all other nodes have been received (cf. Lines 1-3 in Listing 1). In contrast to the NRI, VM heaviness constantly changes and, therefore, each heaviness set that a compute node sends is associated with a certain period of time. Thus, this information needs to be synchronized, when it reaches a compute node. To do so, every compute node n_a maintains a FIFO queue for every other compute node n_b that stores heaviness sets received from n_b. When all of these queues contain at least one element, n_a dequeues the first element from every queue to recalculate the user heaviness and update VM priorities.

E. Calculating and Applying Priorities

The FS allocates resources to VMs by PPs (cf. Lines 10 and 11 of Listing 1). The PP of a VM is a non-negative number for each resource. These ratios of PPs of VMs sharing a resource define the percentage that VMs are allocated of this resource. For example, when two VMs v_1 and v_2 share a resource r and have PPs 1 and 2, respectively, v_1 is allocated $1/3$ of r and v_2 is allocated $2/3$. In case some VMs do not fully utilize their share, the leftover is allocated in the same manner to VMs that request more of this resource. Thus, PPs do not waste resources, since a resource will always be fully allocated, if at least one VM requests it.

The allocation paradigm of PPs is best known as weighted Max-min fairness [25]. Operating systems allow to allocate most time-shared resources by PPs. However, the name to refer to PPs differs depending on the resource: In the context of CPU PPs are termed *shares*, in the context of disk I/O *weights*, and in the context of network access they are associated to certain *queuing disciplines* or *traffic classes*.

All resources except network access are controlled by libvirt, wherefore the FS supports most of the known hypervisors [26]. The FS calculates PPs of a VM v_i based on the number $h_p(v_i) := h_v(v_i) + h_u(o(v_i))$. Number $h_u(o(v_i))$ already includes $h_v(v_i)$ (cf. Equation 4). However, by adding $h_v(v_i)$ again, the individual heaviness of the VM is emphasized, when setting the VM's PPs (otherwise all VMs of a user would have the same PPs). A basic mapping function translates $h_p(v_i)$ to PPs. It is future work to define more elaborated mappings. Generally, the ranges of PPs differ among resources:

CPU time is controlled by setting PPs, alias CPU shares, in the range [1,100].

RAM is space-shared and not time-shared. Thus, it cannot be allocated by PPs. In turn, the FS uses soft limits, which are a minimum guarantee. The FS assigns soft-limits from 10 MiB to the VMs' maximum amount of RAM.

Disk I/O is controlled by setting PPs, alias disk weights, in the range [100,1000]. This is the maximal range allowed by libvirt.

Network access is the only resource that is not controlled by libvirt, as libvirt only allows setting hard limits for network access, *i.e.*, a maximum bandwidth that cannot be exceeded even if no other VM sends data. To avoid the corresponding potential waste of bandwidth, the FS currently deploys the more sophisticated HTB qdisc of `tc` by calling tc's corre-

sponding Command Line Interfaces (CLI). This allows setting PPs in the range [1, 98].

V. Evaluation

The FS is evaluated in terms of CPU overhead and fairness promotion among users. The evaluation environment was set up according to the OpenStack installation guide for Ubuntu 14.04 [20]. All compute nodes are equipped with a 3 GHz dual-core Intel Xeon E3113 CPU, 4 GB RAM, a 150 MiB/s hard-drive, and a 1 Gbit/s network connection.

The efficiency/utilization of the FS is not evaluated, because the mechanism deploys PPs to achieve the runtime prioritization. These PPs ensure that no resource is idle, if desired by a consumer (cf. Section IV-E), and, therefore, guarantee high utilization. Furthermore, the FS either increases or decreases the PPs of a VM on all resources. Thus, no adverse ratios of PPs on different resources occur.

A. CPU Overhead

The CPU overhead is evaluated depending on (i) the number of VMs, (ii) whether or not VMs are loaded, and (iii) the length of the update interval. The evaluation was performed only for a single node, because the current implementation of the FS decentralizes, but not yet optimizes the message exchange between nodes (cf. Section III-E and IV-D1). Accordingly, a *performance experiment* is defined by the triple $(\beta, \lambda, \mu) \in \mathbb{N}_{\geq 1} \times \{T, F\} \times \mathbb{N} \cup \{\infty\}$. β defines the number of VMs that are hosted by the node in the experiment. The Boolean variable λ specifies whether these VMs are loaded, whereat load is simulated by `stress` [1]. μ determines the length of the update interval. If $\mu = \infty$, the FS is deactivated. Each performance experiment runs four minutes. The FS's CPU overhead is measured by the CPU time that is utilized by OpenStack processes, when the FS is running ($\mu < \infty$) or not running ($\mu = \infty$). In particular, the OpenStack processes to be monitored are the `nova-compute`, `nova-network`, `nova-api-metadata`, and `nova-fairness` services.

Figure 1a and 1c show the sum of the CPU time consumed by these processes for different numbers of VMs and update intervals. The figures show that the FS significantly increases the CPU time that is utilized by these processes.

Controlling network access was identified as main cause for the significant CPU overhead. In particular, as outlined in Section IV-E, the Unix application tc is used to apply network priorities, which implies multiple CLI calls to set priorities. Therefore, Figure 1b and 1d illustrate how much CPU time is utilized by OpenStack processes, when the FS does not control the network access. The figures show that disabling network control reduces the CPU overhead by more than 50% on average.

All figures show that the CPU overhead increases linearly with the number of VMs and shorter update intervals. Therefore, the FS's CPU overhead can be reduced by selecting longer update intervals. In particular, the CPU overhead caused by higher numbers of VMs can be contained by increasing μ.

B. Fairness Promotion Among Users

The two *fairness experiments* following show if and how the FS alters VMs' PPs to promote fairness among users. In these fairness experiments the update interval of the FS is 10 seconds and the heaviness metric used by the FS is the greediness metric [24]. All evaluations have been conducted over a timespan of 15 minutes. Although the FS is able to monitor and control CPU time, RAM, disk I/O, and network access, only CPU time and disk I/O are considered in these fairness experiments to simplify the discussion. All users in those experiments have the same quota, wherefore the subtrahend in Equation 4 is the same for all users. Thus, the size of this quota does not influence results of the experiments.

When the fairness service is not running, allocations are determined by the Completely Fair Scheduler (CFS) [**?**], which is the Linux default scheduler and achieves virtually perfect CPU fairness between (VM) processes sharing a node. However, the CFS is oblivious to which cloud user owns a VM process. To the best of the authors' knowledge, the FS is the first implementation that establishes cloud fairness solely by adapting the runtime prioritization of VM processes. In particular, a comparison to DRF or BBF is not possible, because DRF and BBF introduce fairness by VM scheduling but not runtime prioritization.

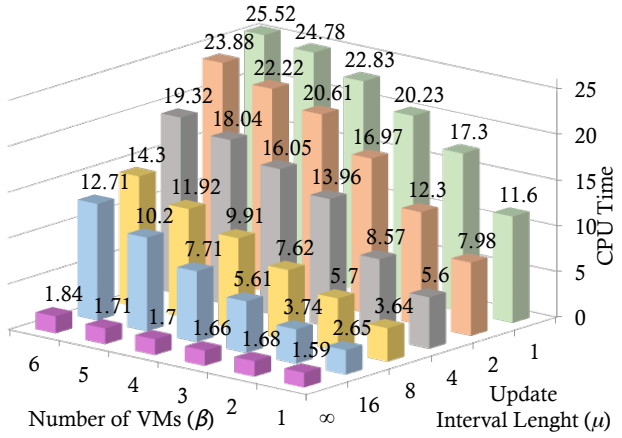
Users and nodes are denoted by u_x and n_y, respectively, and distinguished by subscripts. A VM that belongs to user u_a and is hosted by node n_b is denoted by v_b^a, *i.e.*, VMs are denoted by v and the subscript denotes their host and the superscript their owner.

1) Single node, single resource: The setup of the first fairness experiment is illustrated in Figure 2a and investigates how the FS promotes fairness on one host n_a, when two users u_d and u_e content for the same resource. User u_d operates two VMs $\dot{v}_a^d$ and $\ddot{v}_a^d$ and user u_e operates one VM v_a^e. All VMs attempt to utilize a maximal amount of CPU time, while not imposing significant load on any other resource.

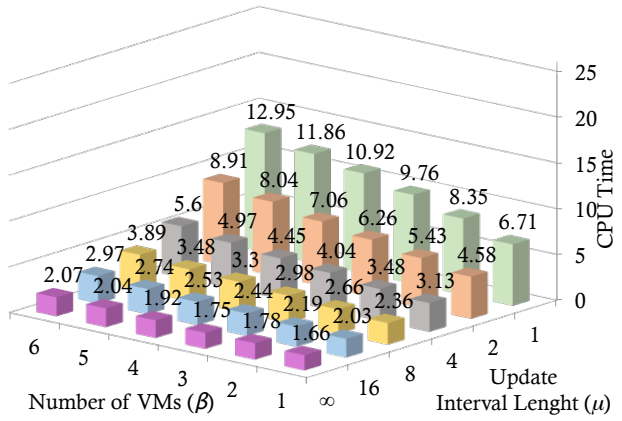
Without the FS all VMs receives $1/3$ of n_a's CPU time, which is arguably not fair, as both users have the same quota but u_d receives twice the amount of CPU time as u_e. Figure 3 shows, how the FS mitigates this unfairness by giving more CPU shares and, thus, more CPU time to the only VM of u_e.

The first three minutes demonstrate the FS's flexibility: Within this timeframe only $\dot{v}_a^d$ attempts to utilizes the CPU, which increases u_d's heaviness and accordingly decreases the CPU shares of u_d's VMs. Nonetheless, $\dot{v}_a^d$ is able to fully utilize the CPU, because no other VM attempts to utilize it. Only after three minutes, when the other VMs also start utilizing the CPU, the shares take effect and u_d's VMs are throttled in favor of u_e's VM.

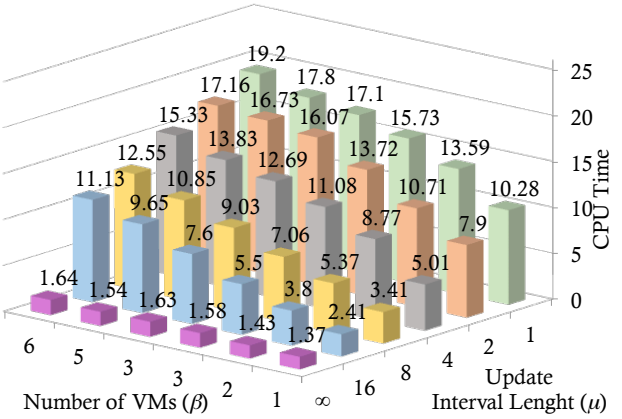
2) Multiple Nodes, Multiple Resources: The second fairness experiment investigates how the FS promotes fairness across nodes, when users content for multiple PRs. Figure 2b shows this setup, where three users u_d, u_e, and u_f utilize PRs on two nodes n_a and n_b. User u_d heavily utilizes the CPU and disk on n_a by VM v_a^d. User u_e attempts the same PR utilization by VM v_a^e. However, u_e additionally utilizes the CPU of host



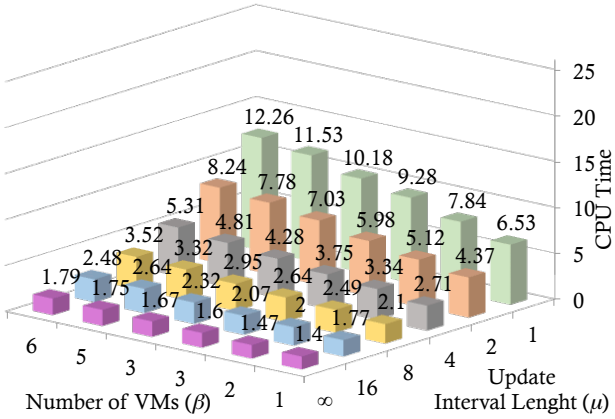
(a) Network is controlled and $\lambda = F$



(b) Network is not controlled and $\lambda = F$

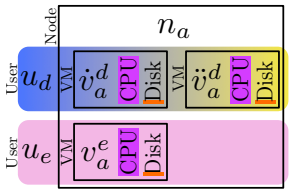


(c) Network is controlled and $\lambda = T$

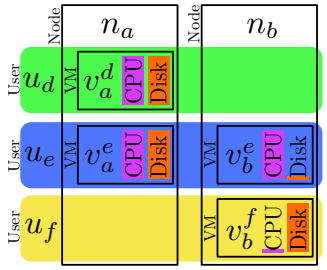


(d) Network is not controlled and $\lambda = T$

Fig. 1: CPU time consumed by OpenStack services dependent on the number of VMs (β) and the update interval (μ)



(a) One node n_a hosts three VMs $\dot{v}_a^d, \ddot{v}_a^d$, and v_a^e, whereat $\dot{v}_a^d, \ddot{v}_a^d$ belong to user u_d and v_a^e belongs to user u_e



(b) Two nodes host four VMs that belong to three users that compete for CPU and disk I/O

Fig. 2: Illustration of the two fairness experiments

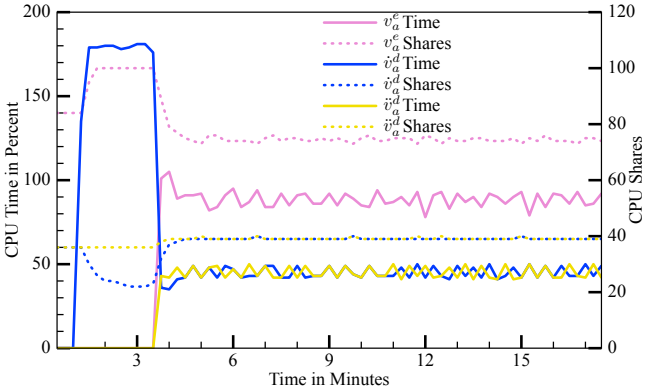
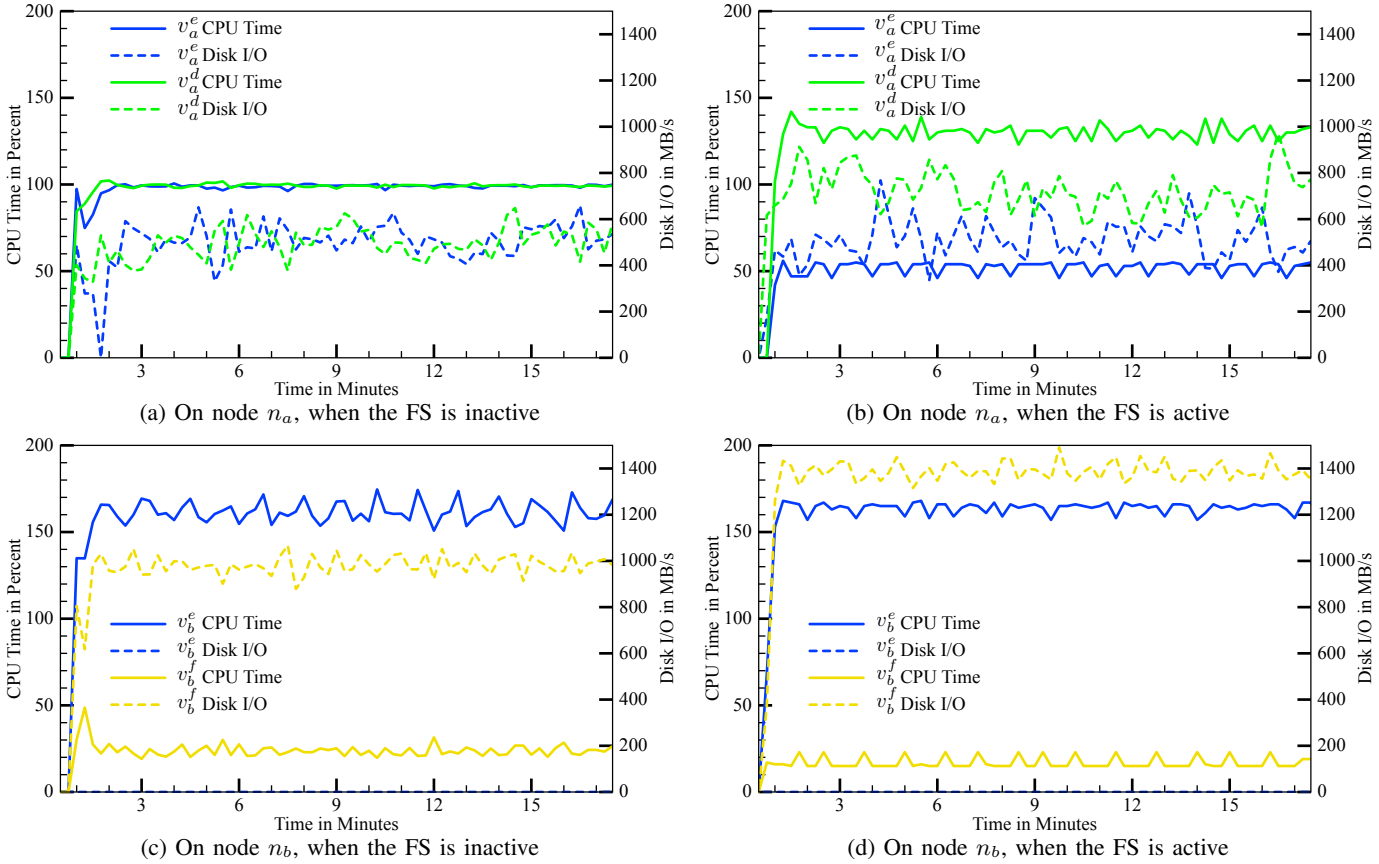


Fig. 3: CPU shares and resulting CPU time allocated by the FS in the fairness experiment as illustrated in Figure 2a

n_b by VM v_b^e. VM v_b^e shares this host with u_f's only VM v_b^f that heavily utilizes disk. Therefore, the utilization of v_b^e and v_b^f on n_b does not interfere, while v_a^d and v_a^e contend for CPU and disk I/O on n_a.

Figure 4 compares CPU and disk allocations with and without the FS on both hosts. The resources user u_e's VMs receive are illustrated by blue lines in all figures. Resources allocated to user u_d and u_f VMs are illustrated by green and yellow lines, respectively.

Figure 4a and 4b compare the allocation on n_a. Figure 4a shows that without the FS both VMs on n_a receive the same amount of CPU and disk I/O. Figure 4b shows that the FS decreases v_a^e's CPU and disk I/O allocation in order to allocate more of these PRs to v_a^d. The reason is that the owner of v_a^e also consumes PRs on n_b. The allocation on n_b is compared by Figure 4c and 4d (v_b^e does not utilize the disk, wherefore the according line is not visible in these figures) and shows that the FS increases v_b^f's disk allocation by almost 40%. The reason is that v_b^e's owner also utilizes PRs on n_a and, therefore,



(a) On node n_a, when the FS is inactive

(b) On node n_a, when the FS is active

(c) On node n_b, when the FS is inactive

(d) On node n_b, when the FS is active

Fig. 4: CPU and disk allocation of the fairness experiment as illustrated in Figure 2b

v_b^f is prioritized. Interestingly, although the FS increases v_b^f's disk allocation by 40%, this does not hurt v_b^e.

3) Findings: Without the FS, OpenStack does not leverage runtime prioritization to promote fairness among cloud users. The simple scenario where two users compete for one resource on the same host, most clearly shows this shortcoming (cf. Section V-B1). Notably, in this scenario fairness can be established without global monitoring information or a multi-resource fairness definition. The FS not only achieves fairness in this scenario, but also in complex settings, where VMs running on different hosts and utilizing different resources have to be managed (cf. Section V-B2).

VI. Conclusions and Future Work

The fairness service presented in this paper successfully adapts PPs of VMs to achieve fairness during VM runtime. The fairness service, therefore, complements other mechanisms that achieve fairness by VM scheduling. The CPU time consumed by the fairness service increases with the number of VMs and shorter update intervals. However, this overhead is mainly caused by the inefficiency of controlling network access by `tc`. The use of `tc` is necessary, as `libvirt`, which is used to control all other resources, only offers hard-limits to control network access.

By handicapping heavy users to prioritize light users, the fairness service enforces an intuitive and practically applicable definition of fairness. To allow the fairness service capturing different notions of fairness, the definition of heaviness is adaptable with a recommended choice being [23], [24]. Guidelines were given, of how to define heaviness, such that users have incentive to configure their VMs correctly and, thereby, help utilizing the cloud most efficiently. Furthermore, it was shown, how to define heaviness to capture DRF and BBF.

Bounds for the message volume produced by the currently fully decentralized message exchange of the fairness service were presented. It was shown that the volume can be reduced by centralization or message exchange techniques from P2P networks. Accordingly, future work is to find a good tradeoff between message volume and the degree of decentralization. Also, an efficient controlling of network access will be investigated. A further planned improvement of the implementation is to account for more resources, such as GPUs, disk space, and software licenses. Additionally, a different metric to normalize CPU time across nodes will be implemented. Finally, the definition of conclusive functions for mapping the heaviness of users to PPs is intended.

References

[1] Amos Waterland. stress. http://people.seas.harvard.edu/~apw/stress/, 2014. Online; accessed February 10th, 2016.

[2] Arka A. Bhattacharya, David Culler, Eric Friedman, Ali Ghodsi, Scott Shenker, and Ion Stoica. Hierarchical Scheduling for Diverse Datacenter

Workloads. *4th Annual Symposium on Cloud Computing*, SOCC'13, pp 1–15, Santa Clara, CA, USA, October 2013.

[3] Thomas Bonald and James Roberts. Enhanced Cluster Computing Performance through Proportional Fairness. *Performance Evaluation*, 79:134–145, April 2014.

[4] Thomas Bonald and James Roberts. Multi-Resource Fairness: Objectives, Algorithms and Performance. *2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'15, pp 31–42, Portland, OR, USA, June 2015.

[5] David Breitgand, Zvi Dubitzky, Amir Epstein, Alex Glikson, and Inbar Shapira. SLA-aware Resource Over-commit in an IaaS Cloud. *8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualization Management (SVM)*, pp 73–81, Las Vegas, NV, USA, October 2012.

[6] Citrix Systems, Inc. AMQP and Nova. http://docs.openstack.org/developer/nova/rpc.html, 2010. Online; accessed February 5th, 2016.

[7] D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, June 2005.

[8] Danny Dolev, Dror G. Feitelson, Joseph Y. Halpern, Raz Kupferman, and Nathan Linial. No Justified Complaints: On Fair Sharing of Multiple Resources. *3rd Innovations in Theoretical Computer Science Conference*, ITCS'12, pp 68–75, Cambridge, MA, USA, January 2012.

[9] Frank Kelly et al. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *The Journal of the Operational Research Society*, 49(3):237–252, March 1998.

[10] Yoav Etsion, Tal Ben-Nun, and Dror G. Feitelson. A Global Scheduling Framework for Virtualization Environments. *2009 IEEE International Symposium on Parallel Distributed Processing*, IPDPS'09, pp 1–8, Rome, Italy, May 2009.

[11] Eric Friedman, Ali Ghodsi, and Christos-Alexandros Psomas. Strategyproof Allocation of Discrete Jobs on Multiple Machines. *15th ACM Conference on Economics and Computation*, EC'14, pp 529–546, Palo Alto, CA, USA, June 2014.

[12] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant Resource Fairness: Fair Allocation of Heterogeneous Resources in Datacenters. Technical Report UCB/EECS-2010-55, EECS Department, University of California, Berkeley, CA, USA, May 2010.

[13] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. Navigating Heterogeneous Processors with Market Mechanisms. *IEEE 19th International Symposium on High Performance Computer Architecture*, HPCA'13, pp 95–106, Shenzhen, China, February 2013.

[14] Avital Gutman and Noam Nisan. Fair Allocation without Trade. *11th International Conference on Autonomous Agents and Multiagent Systems*, Vol. 2 of *AAMAS'12*, pp 719–728, Valencia, Spain, June 2012.

[15] Dalibor Klusáček and Hana Rudová. Multi-resource Aware Fairsharing for Heterogeneous Systems. In: Walfredo Cirne and Narayan Desai (Editors), *18th International Workshop on Job Scheduling Strategies for Parallel Processing, Revised Selected Papers*, JSSPP'14, pp 53–69. Springer International Publishing, May 2015.

[16] Dalibor Klusáček, Hana Rudová, and Michal Jaroš. Multi Resource Fairness: Problems and Challenges. In: Narayan Desai and Walfredo Cirne (Editors), *Job Scheduling Strategies for Parallel Processing*, Vol. 8429 of *Lecture Notes in Computer Science*, pp 81–95. Springer, Berlin/Heidelberg, Germany, 2014.

[17] Haikun Liu and Bingsheng He. Reciprocal Resource Fairness: Towards Cooperative Multiple-Resource Fair Sharing in IaaS Clouds. *2014 IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'14, pp 970–981, New Orleans, LA, USA, November 2014.

[18] Haikun Liu and Bingsheng He. F2C: Enabling Fair and Fine-grained Resource Sharing in Multi-tenant IaaS Clouds. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, November 2015.

[19] OpenStack Foundation. OpenStack Wiki: ZeroMQ. https://wiki.openstack.org/wiki/ZeroMQ, 2014. Online; accessed February 5th, 2016.

[20] OpenStack Foundation. OpenStack Installation Guide for Ubuntu 14.04. http://docs.openstack.org/juno/install-guide/install/apt/content/index.html, 2015. Online; accessed February 4th, 2016.

[21] OpenStack Foundation. Services, Managers and Drivers. http://docs.openstack.org/developer/nova/services.html, 2016. Online; accessed February 5th, 2016.

[22] David C. Parkes, Ariel D. Procaccia, and Nisarg Shah. Beyond Dominant Resource Fairness: Extensions, Limitations, and Indivisibilities. *13th ACM Conference on Electronic Commerce*, EC'12, pp 808–825, Valencia, Spain, June 2012.

[23] Patrick Poullie and Burkhard Stiller. Cloud Flat Rates Enabled via Fair Multi-resource Consumption. Technical Report IFI-2015.03, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/publications/IFI-2015.03.pdf, Universität Zürich, Zurich, Switzerland, October 2015.

[24] Patrick Poullie and Burkhard Stiller. Cloud Flat Rates Enabled via Fair Multi-resource Consumption. *10th International Conference on Autonomous Infrastructure, Management and Security,* AIMS'16, Vol. 9701 of *Lecture Notes in Computer Science*, Munich, Germany, June 2016.

[25] Bozidar Radunovic and Jean-Yves Le Boudec. A Unified Framework for Max-Min and Min-Max Fairness With Applications. *IEEE/ACM Transactions on Networking*, 15(5):1073–1083, October 2007.

[26] Red Hat, Inc. Libvirt: Internal Drivers. https://libvirt.org/drivers.html, 2016. Online; accessed February 5th, 2016.

[27] Redis Labs. Redis. http://redis.io/, 2015. Online; accessed February 5th, 2016.

[28] Standard Performance Evaluation Corporation. Benchmarks. http://spec.org/benchmarks.html, 2015. Online; accessed February 20th, 2016.

[29] Chandra Thimmannagari. *CPU Design: Answers to Frequently Asked Questions*. Springer, Berlin/Heidelberg Germany, 2005.

[30] G. Wei, A. Vasilakos, Y. Zheng, and N. Xiong. A Game-theoretic Method of Fair Resource Allocation for Cloud Computing Services. *The Journal of Supercomputing*, 54(2):252–269, November 2010.

[31] Yoel Zeldes and Dror G. Feitelson. On-line Fair Allocations Based on Bottlenecks and Global Priorities. *4th ACM/SPEC International Conference on Performance Engineering*, ICPE'13, pp 229–240, Prague, Czech Republic, April 2013.

[32] Qinyun Zhu and Jae C. Oh. An Approach to Dominant Resource Fairness in Distributed Environment. In: Moonis Ali, Young Sig Kwon, Chang-Hwan Lee, Juntae Kim, and Yongdai Kim (Editors), *Current Approaches in Applied Artificial Intelligence*, Vol. 9101 of *Lecture Notes in Computer Science*, pp 141–150. Springer International Publishing, 2015.