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 [1]. While in public/commercial clouds resource allocation to VMs is prescribed by Service Level Agreements (SLA) [2], 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 [3].

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 in an integrated manner as presented in all levels of details in [4]. 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

978-3-901882-85-2

VM runtime also allows commercial clouds to introduce much 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 [3].

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. 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 (DRF's application case) ratios in which resources are required may change, this is even more likely, during runtime prioritization (BBF's application case). Thus, this work here distinguishes itself from DRF and BBF, as it (i) addresses the problem of fair runtime prioritization in a manner that is coordinated among nodes, (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.

II. Related Work

Multi-resource Fairness issues in virtualization environments were first pointed out by [5]. As a solution, [5] defines Bottleneck-based Fairness (BBF). [6], [7] provide theoretical insights on this concept and [8] 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 [9]. DRF has been extended in many directions, for example by indivisibilities of resources and user hierarchies [10], [11], [12], [13]. A third approach to data centers fairness is the extension of Proportional Fairness [14] to multiple resources [15], [16]. All three approaches (BBF, DRF, proportional fairness) assume that resources are required in static ratios. Thus, these approaches are not suited to be applied to runtime prioritization of VMs, as here resource dependencies are unpredictable. Only [8] 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. [17], [18] 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.

[19], [20] allows a coalition of users to trade resources and adapt their VM runtime priorities. The adopted fairness notion is asset fairness. [19], [20] requires trading mechanisms and VM demand prediction. When a VM does not utilize all resources it is entitled to based on its configuration, those resources may not always be fully utilized by other VMs. Nonetheless all of these resources are counted, as if they were contributed to other VMs, wherefore, user's have no incentive to configure their VMs correctly.

Thus, those 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 $\mathfrak{o}\colon V \to U$ maps a VM to its owner, *i.e.*, the user that started the VM, and $\mathfrak{a}\colon V \to N$ maps a VM to the node 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 $\mathfrak{r}\colon V \cup N \cup U \to \mathbb{R}^m_{\geq 0}$ maps (i) VMs to their VRs, (ii) nodes to their PRs, and (iii) users to their quota. Function $\mathfrak{u}\colon V \to \mathbb{R}^m_{\geq 0}$ maps VMs to the load they impose on the PRs (at a distinct point in time). Arithmetic operations on vectors are applied point-wise.

Figure 1 illustrates the definitions required. The upper three circles represent the three sets U, V, and N, as explained above. The lower three circles represent the VRs, PRs, and $\mathbb{R}$ sets. Functions are represented by arrows between the sets, *e.g.*, $\mathfrak{r}$ pointing from V to VRs depicts that function $\mathfrak{r}$ maps

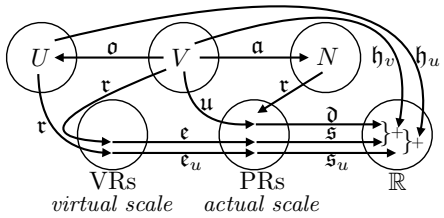


Fig. 1: Dependencies among the definitions of Section III

VMs to their VRs. The endowment function $\mathfrak{e}$ (cf. Equation 1) determines a VM's endowment to PRs based on the VM's VRs. Therefore, $\mathfrak{e}$ connects the $\mathfrak{r}$-arrow to PRs. Note that, although the $\mathfrak{e}$-arrow begins in the set VRs, function $\mathfrak{e}$ actually maps from V. Values in VRs (and PRs) are vectors. Function $\mathfrak{s}$ maps a VM's endowment to a scalar, indicated by the arrow from the end of the $\mathfrak{e}$-arrow to $\mathbb{R}$. $\mathfrak{h}_v$ maps a VM to its heaviness, which is the sum of the VM's $\mathfrak{s}$ and $\mathfrak{d}$ values. Function $\mathfrak{h}_v$ is represented by the arrow labeled $\mathfrak{h}_v$, which points from the set VMs to the plus between the ends of the $\mathfrak{s}$ and $\mathfrak{d}$ arrows.

A. Problem Statement

Cloud runtime fairness cannot be uniquely defined [3]. 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 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 $\mathfrak{h}_u\colon U \to \mathbb{R}$ 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 $\mathfrak{h}_v\colon V \to \mathbb{R}$. Metric $\mathfrak{h}_v$ can be defined by a function $\mathfrak{d}\colon \mathbb{R}^m_{\geq 0} \to \mathbb{R}$ that maps the VM's multi-resource load vector to a heaviness scalar, as for example the functions presented in [18], [17] or functions that define the value of a vector as asset fairness or DRF do (see [4] for how to define $\mathfrak{h}_u$ 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, $\mathfrak{h}_u$ 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 when it is idle (a VM is idle, when it is running but does not execute any workload). u_a has not instantiated any other VMs. 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, wherefore 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 $\mathfrak{h}_v$ 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 $\mathfrak{h}_u$ 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 $\mathfrak{u}$) 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. To make different resources comparable, a resource unit must be divided by the cloud-wide supply of that resource. The cloud-wide supply of every resources is contained in the

Cloud Resource Supply (CRS) $:= \sum_{n_j \in N} \mathfrak{r}(n_j)$. To calculate the CRS, 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 $\mathfrak{h}_u$ 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 $\mathfrak{e}\colon V \to \mathbb{R}^m_{\geq 0}$ to calculate the endowment. [4] presents three different functions to define $\mathfrak{e}$ and identifies

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

as the best choice. As the host of a VM is chosen based on how many PRs are budgeted for this VM, this definition defines the endowment of a VM as the share of the VM's host's PRs that is proportional to the VM's VRs. While VRs are on the virtual scale, this endowment definition maps VRs to the actual scale.

4) VM Heaviness: The heaviness of a VM v_i is defined by $\mathfrak{h}_v(v_i) := \mathfrak{s}(\mathfrak{e}(v_i)) + \mathfrak{d}(v_i)$. Function $\mathfrak{s}\colon \mathbb{R}^m_{\geq 0} \to \mathbb{R}_{\geq 0}$ defines the *static heaviness* to introduce a cost for instantiating a VM irrespective of its load only based on its VRs. Function $\mathfrak{s}$ 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, $\mathfrak{s}$ must strictly increase with each input dimension.

Function $\mathfrak{d}\colon V \to \mathbb{R}$ 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 $\mathfrak{u}(v_i) = \mathfrak{u}(v_j) = x + y$, $\mathfrak{e}(v_i) = x$ and, $\mathfrak{e}(v_j) = x + y$, *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 $\mathfrak{s}(v_j) > \mathfrak{s}(v_i)$ holds true. It is crucial to choose functions $\mathfrak{s}$ and $\mathfrak{d}$, such that $\mathfrak{h}_v(v_i) > \mathfrak{h}_v(v_j)$, 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 $\mathfrak{d}(v_i) > -\mathfrak{s}(\mathfrak{e}(v_i))$. In particular, one may wish to define the dynamic heaviness such that it is zero, when $\mathfrak{u}(v_i) = \mathfrak{e}(v_i)$, 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 $\mathfrak{d}$ and $\mathfrak{s}$, such that $\mathfrak{d}(v_i) > -\mathfrak{s}(\mathfrak{e}(v_i))$ and the non-minimalistic condition hold, the configuration and utilization incentive are provided (cf. [21] for an example).

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 *user endowment* 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

$$\mathfrak{e}_u(u_i) \mapsto \frac{\sum_{n_j \in N} \mathfrak{r}(n_j)}{\sum_{u_k \in U} \mathfrak{r}(u_k)} \cdot \mathfrak{r}(u_i). \tag{2}$$

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

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.*,

$$\mathfrak{h}_u(u_i) := \left(\sum_{v_j \in V : \mathfrak{o}(v_j) = u_i} \mathfrak{h}_v(v_j) \right) - \mathfrak{s}_u(\mathfrak{e}_u(u_i)). \tag{3}$$

Although BBF and DRF are complementary to the fairness approach taken in this paper, both fairness notions include a metric to quantify what users receive. [4] discusses how

these metrics can be reformulated to define $\mathfrak{h}_u$ and why it is not an optimal choice. In contrast, [3] develops a well suited definition of $\mathfrak{h}_u$ based on a survey among more than 600 individuals.

IV. Implementation

Out of the box, OpenStack does not leverage runtime prioritization to manage resource allocation. Thus, the extension of OpenStack was performed to introduce the Fairness Service (FS) being implemented to enforce fairness as discussed.

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.

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 $\mathfrak{d}, \mathfrak{e}, \mathfrak{s}$, and $\mathfrak{s}'$ in oder to adapt the heaviness definition $\mathfrak{h}_u$ (cf. Equation 3). 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 node 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 [22], measures CPU usage. CPU time provided by different nodes is not directly comparable. The reason is that cloud nodes are rarely homogeneous [23] 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 powerful node. To compare CPU time across nodes, the FS normalizes CPU time by the nodes' BogoMIPS [24]. 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 [25], are considered for future improvements.

2) Resource Utilization Information: 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 utilization since boot time. Therefore, the FS calculates the RUI for the current update interval by subtracting the accumulated utilization at the beginning of the interval from the accumulated utilization 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 utilization in the RUI vector.

C. Heaviness Metric

The generic heaviness metric $\mathfrak{h}_u$ presented in Section III-C contains wildcard functions $\mathfrak{d}, \mathfrak{e}, \mathfrak{s}$ and $\mathfrak{s}'$. 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 definition for function $\mathfrak{e}$ in Section III-C3 assumes 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 node 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. By default, the message exchange among compute nodes is centralized, as `RabbitMQ` [27] is used. The use of `ZeroMQ` [28] decentralizes the information exchange with minimal configuration changes. In both cases, the message volume is quadratic in the number of compute nodes, as every node sends messages to every other compute node. Therefore, a message scheme was designed, that arranges the information flow among compute nodes as a ring, *i.e.*, every compute node sends messages to and receives messages from exactly one other compute node. The size of the messages send on this ring is linear in the number of users. Therefore, this scheme, is scalable and puts minimal workload on individual nodes.

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 [29]. 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 $h_p(v_i) := \mathfrak{h}_v(v_i) + \mathfrak{h}_u(\mathfrak{o}(v_i))$. Number $\mathfrak{h}_u(\mathfrak{o}(v_i))$ already includes $\mathfrak{h}_v(v_i)$ (cf. Equation 3). 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 produces traffic. To avoid the corresponding potential waste of bandwidth, the FS currently deploys the more sophisticated HTB qdisc of `tc` by calling tc's corresponding Command Line Interfaces (CLI). This allows setting PPs in the range [1, 98].

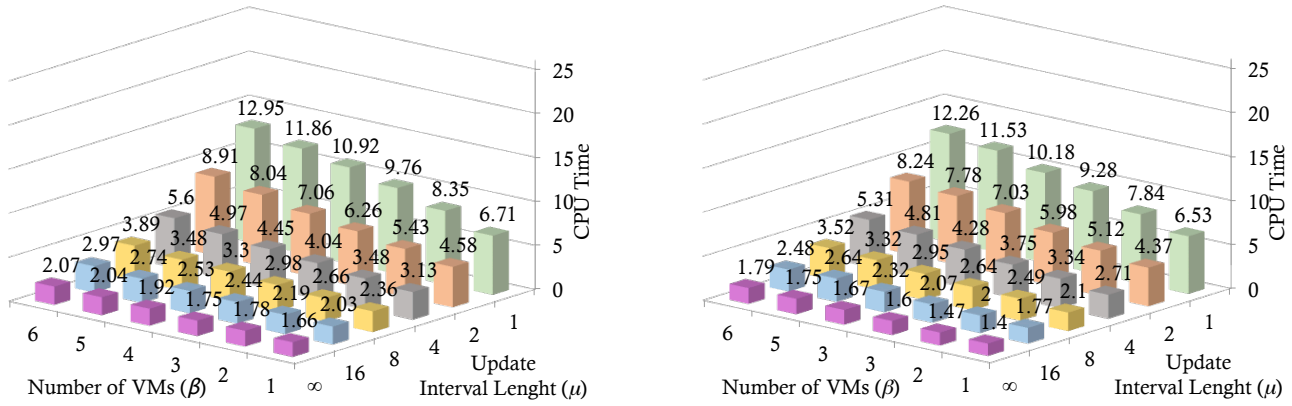
V. Evaluation

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 [30]. 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 FS 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.

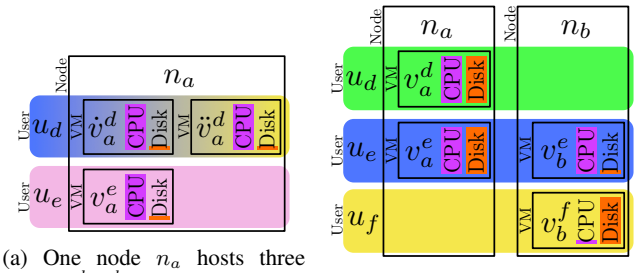
A. CPU Overhead

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 for a single node, as the message exchange among nodes can be implemented, such that the number of messages a node receives and sends is a small constant independent of the overall number of nodes (cf. Section IV-D). Therefore, the number of nodes does not influence the CPU overhead. 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` [31]. 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.



(a) $\lambda = F$ (b) $\lambda = T$

Fig. 2: 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. 3: Illustration of the two fairness experiments

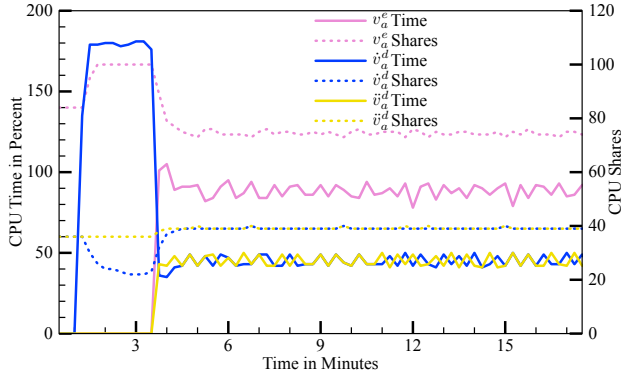


Fig. 4: CPU shares and resulting CPU time allocated by the FS in the fairness experiment as illustrated in Figure 3a

Controlling network access was identified to cause 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 2 illustrates how much CPU time is utilized by OpenStack processes, when the FS does not control the network access. [4] presents equivalent figures showing how much CPU time is utilized, when the network access is controlled.

Figure 2 shows 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.

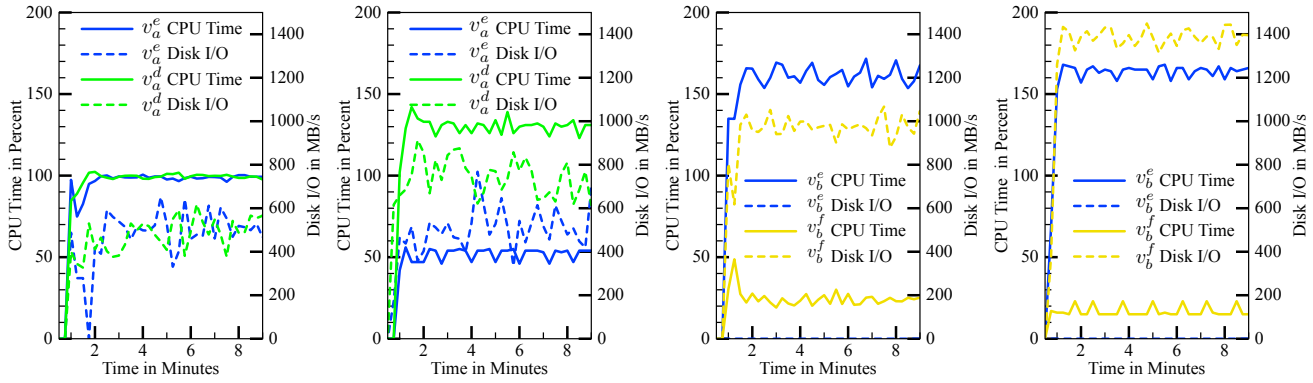
B. Fairness Promotion Among Users

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 [3]. [32], [21] include exhaustive numerical evaluations of using the greediness metric to prioritize VMs. 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 3 is the same for all users. Thus, the size of this quota does not influence results of the experiments.

When the FS is not running, allocations are determined by the Completely Fair Scheduler (CFS) [33], 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.

1) Single node, single resource: The setup of the first fairness experiment is illustrated in Figure 3a and investigates how the FS promotes fairness on one node 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



(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. 5: CPU and disk allocation for the fairness experiment as illustrated in Figure 3b

4 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 3b 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 node n_b by VM v_b^e. VM v_b^e shares this node 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 5 compares CPU and disk allocations with and without the FS on both nodes. 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 5a and 5b compare the allocation on n_a. Figure 5a shows that without the FS both VMs on n_a receive the same amount of CPU and disk I/O. Figure 5b 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 5c and 5d (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, 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 node, 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 nodes and utilizing different resources have to be managed (cf. Section V-B2).

VI. Conclusions and Future Work

This paper proposed a novel and intuitive definition of cloud fairness and extended the OpenStack implementation by a Fairness Service (FS) proving its practical applicability.

The FS complements mechanisms that achieve fairness by VM scheduling, because it adapts VMs' PPs during runtime. VM scheduling only allows for managing a cloud's resource allocation, when VMs are started. Thus, it is ineffective, when VMs run over longer periods of time. Accordingly, the FS is particularly well suited to ensure fairness among users that run VMs over longer periods. Since resource allocations in private clouds are not guided by SLAs, as it is the case in commercial clouds, fairness and, therefore, the FS is more important in private clouds. The resources a cloud user effectively utilizes depend on the load of the user's VMs. Accordingly, even when users have instantiated the same VMs, resources the VMs effectively utilize can be different. Contrary to scheduling schemes, this new FS allows for managing and, thus, streamlining this amount of effectively utilized resources and, therefore, can be deployed to define fair cloud flat rate payment schemes, too [3].

Future work needs to implement a compact message scheme to exchange information of the FS between compute nodes. Another planned improvement is to account for additional resources, such as GPUs, disk space, and software licenses. Finally, the definition of conclusive functions for mapping the heaviness of users to PPs will be undertaken.

References

[1] D. Clark, J. Wroclawski, K. Sollins, and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet," *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 462–475, Jun. 2005. [Online]. Available: http://dx.doi.org/10.1109/TNET.2005.850224

[2] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira, "SLA-aware Resource Over-commit in an IaaS Cloud," in *8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualization Management (SVM)*, Las Vegas, NV, USA, October 2012, pp. 73–81.

[3] P. Poullie and B. Stiller, "Cloud Flat Rates Enabled via Fair Multi-resource Consumption," in *10th International Conference on Autonomous Infrastructure, Management and Security,* AIMS'16, ser. Lecture Notes in Computer Science, vol. 9701, June 2016.

[4] P. Poullie, S. Mannhart, and B. Stiller, "Defining and Enforcing Fairness Among Cloud Users by Adapting Virtual Machine Priorities During Runtime," Universität Zürich, Zurich, Switzerland, Technical Report IFI-2016.04, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/publications/IFI-2016.04.pdf, March 2016.

[5] Y. Etsion, T. Ben-Nun, and D. G. Feitelson, "A Global Scheduling Framework for Virtualization Environments," in *2009 IEEE International Symposium on Parallel Distributed Processing*, ser. IPDPS'09, May 2009, pp. 1–8.

[6] D. Dolev, D. G. Feitelson, J. Y. Halpern, R. Kupferman, and N. Linial, "No Justified Complaints: On Fair Sharing of Multiple Resources," in *3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS'12, Cambridge, MA, USA, January 2012, pp. 68–75.

[7] A. Gutman and N. Nisan, "Fair Allocation without Trade," in *11th International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS'12, vol. 2, Valencia, Spain, June 2012, pp. 719–728.

[8] Y. Zeldes and D. G. Feitelson, "On-line Fair Allocations Based on Bottlenecks and Global Priorities," in *4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE'13, New York, NY, USA, April 2013, pp. 229–240. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479904

[9] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Heterogeneous Resources in Datacenters," EECS Department, University of California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2010-55, May 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-55.html

[10] D. C. Parkes, A. D. Procaccia, and N. Shah, "Beyond Dominant Resource Fairness: Extensions, Limitations, and Indivisibilities." in *13th ACM Conference on Electronic Commerce*, ser. EC'12, June 2012, pp. 808–825.

[11] E. Friedman, A. Ghodsi, and C.-A. Psomas, "Strategyproof Allocation of Discrete Jobs on Multiple Machines," in *15th ACM Conference on Economics and Computation*, ser. EC'14, New York, NY, USA, June 2014, pp. 529–546. [Online]. Available: http://doi.acm.org/10.1145/2600057.2602889

[12] Q. Zhu and J. C. Oh, "An Approach to Dominant Resource Fairness in Distributed Environment," in *Current Approaches in Applied Artificial Intelligence*, ser. Lecture Notes in Computer Science, M. Ali, Y. S. Kwon, C.-H. Lee, J. Kim, and Y. Kim, Eds. Springer International Publishing, 2015, vol. 9101, pp. 141–150.

[13] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical Scheduling for Diverse Datacenter Workloads," in *4th Annual Symposium on Cloud Computing*, ser. SOCC'13, New York, NY, USA, October 2013, pp. 1–15. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523637

[14] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *The Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, March 1998. [Online]. Available: http://www.jstor.org/stable/3010473

[15] T. Bonald and J. Roberts, "Enhanced Cluster Computing Performance through Proportional Fairness," *Performance Evaluation*, vol. 79, pp. 134–145, April 2014. [Online]. Available: https://hal-institut-mines-telecom.archives-ouvertes.fr/hal-01112964

[16] ——, "Multi-Resource Fairness: Objectives, Algorithms and Performance," in *2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS'15, New York, NY, USA, June 2015, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/2745844.2745869

[17] D. Klusáček, H. Rudová, and M. Jaroš, "Multi Resource Fairness: Problems and Challenges," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, N. Desai and W. Cirne, Eds. Berlin/Heidelberg, Germany: Springer, 2014, vol. 8429, pp. 81–95. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43779-7_5

[18] D. Klusáček and H. Rudová, "Multi-resource Aware Fairsharing for Heterogeneous Systems," in *18th International Workshop on Job Scheduling Strategies for Parallel Processing, Revised Selected Papers*, ser. JSSPP'14, W. Cirne and N. Desai, Eds. Springer International Publishing, May 2015, pp. 53–69. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-15789-4_4

[19] H. Liu and B. He, "Reciprocal Resource Fairness: Towards Cooperative Multiple-Resource Fair Sharing in IaaS Clouds," in *2014 IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'14, November 2014, pp. 970–981.

[20] ——, "F2C: Enabling Fair and Fine-grained Resource Sharing in Multi-tenant IaaS Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, November 2015.

[21] P. Poullie and B. Stiller, "The Design and Evaluation of a Heaviness Metric for Cloud Fairness and Correct Virtual Machine Configurations," in *Proceedings of the 13th on Conference on the Economics of Grids, Clouds, Systems, and Services*, ser. GECON'16, September 2016.

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

[23] M. Guevara, B. Lubin, and B. C. Lee, "Navigating Heterogeneous Processors with Market Mechanisms," in *IEEE 19th International Symposium on High Performance Computer Architecture*, ser. HPCA'13, February 2013, pp. 95–106.

[24] Wim van Dorst, "BogoMips mini-HowTo," http://www.clifton.nl/index.html?bogomips.html, 2006, online; accessed February 6th, 2016.

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

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

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

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

[29] B. Radunovic and J.-Y. L. Boudec, "A Unified Framework for Max-Min and Min-Max Fairness With Applications," *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1073–1083, October 2007.

[30] 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.

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

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

[33] M. Tim Jones, "Inside the Linux 2.6 Completely Fair Scheduler," IBM developerWorks, December 2009, http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/; last accessed May 11, 2016.