HUG: Multi-Resource Fairness for Correlated and Elastic Demands

Mosharaf Chowdhury, *University of Michigan;* **Zhenhua Liu,** *Stony Brook University;*
Ali Ghodsi and Ion Stoica, *University of California, Berkeley, and Databricks Inc.*

https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/chowdhury

This paper is included in the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).

March 16–18, 2016 • Santa Clara, CA, USA

ISBN 978-1-931971-29-4

Open access to the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) is sponsored by USENIX.

HUG: Multi-Resource Fairness for Correlated and Elastic Demands

Mosharaf Chowdhury[1], Zhenhua Liu[2], Ali Ghodsi[3], Ion Stoica[3]
[1]*University of Michigan* [2]*Stony Brook University* [3]*UC Berkeley, Databricks Inc.*

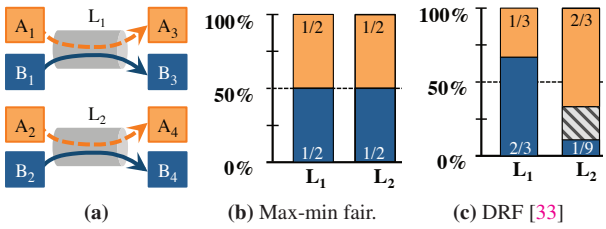
Abstract

In this paper, we study how to optimally provide isolation guarantees in multi-resource environments, such as public clouds, where a tenant's demands on different resources (links) are *correlated*. Unlike prior work such as Dominant Resource Fairness (DRF) that assumes static and fixed demands, we consider *elastic* demands. Our approach generalizes canonical max-min fairness to the multi-resource setting with correlated demands, and extends DRF to elastic demands. We consider two natural optimization objectives: isolation guarantee from a tenant's viewpoint and system utilization (work conservation) from an operator's perspective. We prove that in non-cooperative environments like public cloud networks, there is a strong tradeoff between optimal isolation guarantee and work conservation when demands are elastic. Even worse, work conservation can even decrease network utilization instead of improving it when demands are inelastic. We identify the root cause behind the tradeoff and present a provably optimal allocation algorithm, *High Utilization with Guarantees* (HUG), to achieve maximum attainable network utilization without sacrificing the optimal isolation guarantee, strategyproofness, and other useful properties of DRF. In cooperative environments like private datacenter networks, HUG achieves both the optimal isolation guarantee and work conservation. Analyses, simulations, and experiments show that HUG provides better isolation guarantees, higher system utilization, and better tenant-level performance than its counterparts.

1 Introduction

In shared, multi-tenant environments such as public clouds [2, 5, 6, 8, 40], the need for predictability and the means to achieve it remain a constant source of discussion [15, 44, 45, 50, 51, 58, 59, 61, 62]. The general consensus – recently summarized by Mogul and Popa [53] – is that tenants expect guaranteed minimum bandwidth (i.e., *isolation guarantee*) for performance predictability, while network operators strive for *work conservation* to achieve high utilization and *strategy-proofness* to ensure isolation.

Max-min fairness [43] – a widely-used [16, 25, 34, 35, 55, 63, 64] allocation policy – achieves all three in the context of a single link. It provides the *optimal* isolation guarantee by maximizing the minimum amount of bandwidth allocated to each flow. The bandwidth allocation of a user (tenant) determines her progress – i.e., how fast she can complete her data transfer. It is work-conserving, because, given enough demand, it allocates the entire bandwidth of the link. Finally, it is strategyproof, because tenants cannot get more bandwidth by lying about their demands (e.g., by sending more traffic).



(a) (b) Max-min fair. (c) DRF [33]

Figure 1: Bandwidth allocations in two independent links (a) for tenant-A (orange) with correlation vector $\overrightarrow{d_A} = \langle \frac{1}{2}, 1 \rangle$ and tenant-B (dark blue) with $\overrightarrow{d_B} = \langle 1, \frac{1}{6} \rangle$. Shaded portions are not allocated to any tenant.

However, a datacenter network involves many links, and tenants' demands on different links are often *correlated*. Informally, we say that the demands of a tenant on two links i and j are correlated, if for every bit the tenant sends on link-i, she sends at least α bits on link-j. More formally, with every tenant-k, we associate a correlation vector $\overrightarrow{d_k} = \langle d_k^1, d_k^2, \ldots, d_k^n \rangle$, where $d_k^i \leq 1$, which captures the fact that for every d_k^i bits tenant-k sends on link-i, it should send at least d_k^j bits on link-j.

Examples of applications with *correlated demands* include optimal shuffle schedules [22, 23], long-running services [19, 52], multi-tiered enterprise applications [39], and realtime streaming applications [10, 69]. Consider the example in Figure 1a with two independent links and two tenants. The correlation vector $\overrightarrow{d_A} = \langle \frac{1}{2}, 1 \rangle$ means that (i) link-2 is tenant-A's bottleneck, (ii) for every $\mathcal{M}_A$ rate tenant-A is allocated on the bottleneck link, she requires *at least* $\mathcal{M}_A/2$ rate on link-1, resulting in a progress of $\mathcal{M}_A$, and (iii) except for the bottleneck link, tenants' demands are elastic, meaning tenant-A can use more than $\mathcal{M}_A/2$ rate on link-1.[1] Similarly, tenant-B requires *at least* $\mathcal{M}_B/6$ on link-2 for $\mathcal{M}_B$ on link-1. If we denote the rate allocated to tenant-k on link-i by a_k^i, then $\mathcal{M}_k = \min_i \left\{ \frac{a_k^i}{d_k^i} \right\}$, the minimum demand-normalized rate allocation over all links, captures her progress.

In this paper, we want *to generalize max-min fairness to tenants with correlated and elastic demands while maintaining its desirable properties: optimal isolation guarantee, high utilization, and strategy-proofness.*

[1] While it does not improve the instantaneous progress of tenant-A, it increases network utilization, which is desired by the operators.

Intuitively, we want to maximize the minimum progress over all tenants, i.e., **maximize** $\min_k \mathcal{M}_k$, where $\min_k \mathcal{M}_k$ corresponds to the isolation guarantee of an allocation algorithm. We make three observations. First, when there is a single link in the system, this model trivially reduces to max-min fairness. Second, getting more aggregate bandwidth is not always better. For tenant-A in the example, $\langle 50\text{Mbps}, 100\text{Mbps}\rangle$ is better than $\langle 90\text{Mbps}, 90\text{Mbps}\rangle$ or $\langle 25\text{Mbps}, 200\text{Mbps}\rangle$, even though the latter ones have more bandwidth in total. Third, simply applying max-min fairness to individual links is not enough. In our example, max-min fairness allocates equal resources to both tenants on both links, resulting in allocations $\langle \frac{1}{2}, \frac{1}{2}\rangle$ on both links (Figure 1b). Corresponding progress ($\mathcal{M}_A = \mathcal{M}_B = \frac{1}{2}$) result in a suboptimal isolation guarantee ($\min\{\mathcal{M}_A, \mathcal{M}_B\} = \frac{1}{2}$).

Dominant Resource Fairness (DRF) [33] extends max-min fairness to multiple resources and prevents such suboptimality. It equalizes the shares of dominant resources – link-2 (link-1) for tenant-A (tenant-B) – across *all* tenants with correlated demands and maximizes the isolation guarantee in a strategyproof manner. As shown in Figure 1c, using DRF, both tenants have the *same* progress – $\mathcal{M}_A = \mathcal{M}_B = \frac{2}{3}$, 50% higher than using max-min fairness on individual links. Moreover, DRF's isolation guarantee ($\min\{\mathcal{M}_A, \mathcal{M}_B\} = \frac{2}{3}$) is optimal across all possible allocations and is strategyproof.

However, DRF assumes *inelastic* demands [40], and it is not work-conserving. For example, bandwidth on link-2 in shades is not allocated to either tenant. In fact, we show that DRF can result in *arbitrarily low utilization* (Lemma 6). This is wasteful, because unused bandwidth cannot be recovered.

We start by showing that strategy-proofness is a *necessary* condition for providing the optimal isolation guarantee – i.e., to **maximize** $\min_k \mathcal{M}_k$ – in non-cooperative environments (§2). Next, we prove that work conservation – i.e., when tenants are allowed to use unallocated resources, such as the shaded area in Figure 1c, without constraints – spurs a race to the bottom. It incentivizes each tenant to continuously lie about her demand correlations, and in the process, it decreases the amount of useful work done by all tenants! Meaning, simply making DRF work-conserving can do more harm than good.

We propose a two-stage algorithm, *High Utilization with Guarantees* (HUG), to achieve our goals (§3). Figure 2 surveys the design space for cloud network sharing and places HUG in context by following the thick lines. At the highest level, unlike many alternatives [13, 14, 37, 44], HUG is a dynamic allocation algorithm. Next, HUG enforces its allocations at the tenant-/network-level, because flow- or (virtual) machine-level allocations [61, 62] do not provide isolation guarantee.

Due to the hard tradeoff between optimal isolation guarantee and work conservation in non-cooperative environments, *HUG ensures the highest utilization possible while maintaining the optimal isolation guarantee*. It incentivizes tenants to expose their true demands, ensuring that they actually consume their allocations instead of causing collateral damages. In cooperative environments, where strategy-proofness might be a non-requirement, *HUG simultaneously ensures both work conservation and the optimal isolation guarantee*. In contrast, existing solutions [33, 45, 51, 58, 59] are suboptimal in both environments. Overall, HUG generalizes single- [25, 43, 55] and multi-resource max-min fairness [27, 33, 38, 56] and multi-tenant network sharing solutions [45, 51, 58, 59, 61, 62] under a unifying framework.

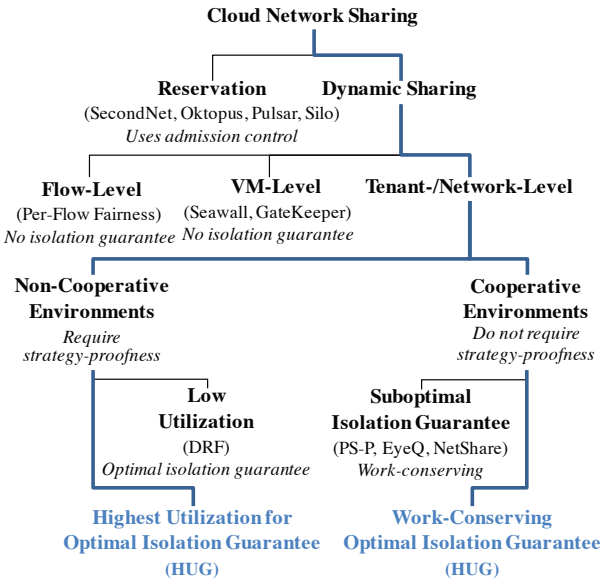


Figure 2: Design space for cloud network sharing.

HUG is easy to implement and scales well. Even with $100,000$ machines, new allocations can be centrally calculated and distributed throughout the network in less than a second – faster than that suggested in the literature [13]. Moreover, each machine can locally enforce HUG-calculated allocations using existing traffic control tools without any changes to the network (§4).

We demonstrate the effectiveness of our proposal using EC2 experiments and trace-driven simulations (§5). In non-cooperative environments, HUG provides the optimal isolation guarantee, which is $7.4\times$ higher than existing network sharing solutions like PS-P [45, 58, 59] and $7000\times$ higher than traditional per-flow fairness, and $1.4\times$ better utilization than DRF for production traces. In cooperative environments, HUG outperforms PS-P and per-flow fairness by $1.48\times$ and $17.35\times$ in terms of the 95th percentile slowdown of job communication stages, and 70% jobs experience lower slowdown w.r.t. DRF.

We discuss current limitations and future research in Section 6 and compare HUG to related work in Section 7.

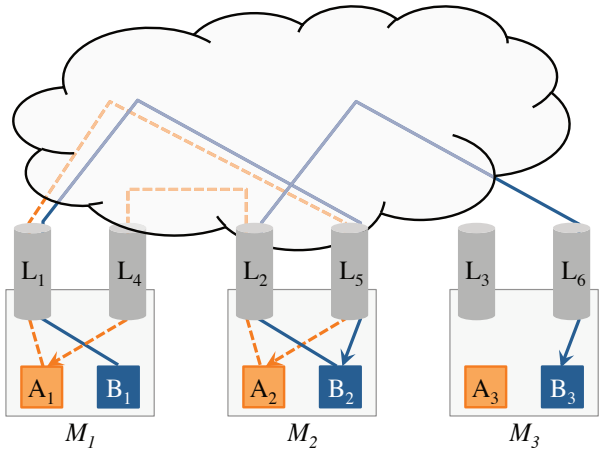


Figure 3: Two VMs from tenant-A (orange) and three from tenant-B (dark blue) and their communication patterns over a 3×3 datacenter fabric. The network fabric has three uplinks (L_1–L_3) and three downlinks (L_4–L_6) corresponding to the three physical machines.

2 Motivation

In this section, we elaborate on the assumptions and notations used in this paper and summarize the three desirable requirements – optimal isolation guarantee, high utilization, proportionality – for bandwidth allocation across multiple tenants. Later, we show the tradeoff between optimal isolation guarantee and high utilization, identifying work conservation as the root cause.

2.1 Background

We consider Infrastructure-as-a-Service (e.g., EC2 [2], Azure [8], and Google Compute [5]) and Container-as-a-Service (e.g., Mesos [40] and Kubernetes [6]) models where tenants pay per-hour flat rates for virtual machines (VMs) and containers.[2]

We abstract out the datacenter network as a non-blocking switch (i.e., the fabric/hose model [11, 12, 14, 23, 28, 45, 49]) with P physical machines connected to it. Each machine has full-duplex links (i.e., $2P$ independent links) and can host one or more VMs from different tenants. Figure 3 shows an example. We assume that VM placement and routing are implemented independently. Not only does this model provide analytical simplicity, it is mostly a reality today: recent EC2 and Google datacenters have full bisection bandwidth networks [4, 7].

We denote the correlation vector of the k-th tenant ($k \in \{1, ..., M\}$) as $\overrightarrow{d_k} = \langle d_k^1, d_k^2, \dots d_k^{2P} \rangle$, where d_k^i and d_k^{P+i} ($1 \le i \le P$) respectively denote the uplink and downlink demands normalized[3] by link capacities (C^i) and $\sum_{i=1}^{P} d_k^i = \sum_{i=1}^{P} d_k^{P+i}$.

[2]We use the terms VM and container interchangeably in this paper because they are similar from the network's perspective.

[3]Normalization helps us consider heterogeneous capacities. By default we normalize the correlation vector such that the largest component equals to 1 unless otherwise specified.

$\overrightarrow{d_k}$	Correlation vector of tenant-k's demand
$\overrightarrow{a_k}$	Guaranteed allocation to tenant-k
$\mathcal{M}_k$	Progress of tenant-k; $\mathcal{M}_k := \min_{1 \le i \le 2P} \left\{ \frac{a_k^i}{d_k^i} \right\}$, where subscript i stands for link-i
$\overrightarrow{c_k}$	Actual resource consumption of tenant-k

Isolation Guarantee	$\min_k \mathcal{M}_k$
Optimal Isolation Guarantee	$\max \{\min_k \mathcal{M}_k\}$
(Network) Utilization	$\sum_i \sum_k c_k^i$

Table 1: Important notations and definitions.

For the example in Figure 3, consider tenant correlation vectors:

$$\overrightarrow{d_A} = \langle \frac{1}{2}, 1, 0, 1, \frac{1}{2}, 0 \rangle$$
$$\overrightarrow{d_B} = \langle 1, \frac{1}{6}, 0, 0, 1, \frac{1}{6} \rangle$$

where $d_k^i = 0$ indicates the absence of a VM and $d_k^i = 1$ indicates the bottleneck link(s) of a tenant.

Correlation vectors depend on tenant applications, that can range from *elastic-demand* batch jobs [3, 24, 42, 68] to long-running services [19, 52], multi-tiered enterprise applications [39], and realtime streaming applications [9, 69] with *inelastic demands*. We focus on scenarios where a tenant's demand changes at the timescale of seconds or longer [13, 18, 58], and she can use provider-allocated resources in any way for her own workloads.

2.2 Inter-Tenant Network Sharing Requirements

Given correlation vectors of M tenants, a cloud provider must use an allocation algorithm $\mathcal{A}$ to determine the allocations of each tenant:

$$\mathcal{A}(\{\overrightarrow{d_1}, \overrightarrow{d_2}, \dots, \overrightarrow{d_M}\}) = \{\overrightarrow{a_1}, \overrightarrow{a_2}, \dots, \overrightarrow{a_M}\}$$

where $\overrightarrow{a_k} = \langle a_k^1, a_k^2, \dots a_k^{2P} \rangle$ and a_k^i is the fraction of link-i guaranteed to the k-th tenant.

As identified in previous work [15, 58], any allocation policy $\mathcal{A}$ must meet three requirements – (optimal) isolation guarantee, high utilization, and proportionality – to fairly share the cloud network:

1. **Isolation Guarantee:** VMs should receive minimum bandwidth guarantees *proportional* to their correlation vectors so that tenants can estimate *worst-case* performance. Formally, *progress* of tenant-k ($\mathcal{M}_k$) is defined as her minimum demand satisfaction ratio across the entire fabric:

$$\mathcal{M}_k = \min_{1 \le i \le 2P} \left\{ \frac{a_k^i}{d_k^i} \right\}$$

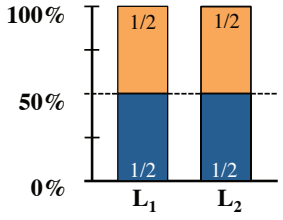


Figure 4: Bandwidth consumptions of tenant-A (orange) and tenant-B (dark blue) with correlation vectors $\overrightarrow{d_A} = \langle \frac{1}{2}, 1 \rangle$ and $\overrightarrow{d_B} = \langle 1, \frac{1}{6} \rangle$ using PS-P [45, 58, 59]. Both tenants run elastic-demand applications.

For example, progress of tenants A and B in Figure 4 are $\mathcal{M}_A = \mathcal{M}_B = \frac{1}{2}$.[4] Note that $\mathcal{M}_k = \frac{1}{M}$ if $\overrightarrow{d_k} = \langle 1, 1, \ldots, 1 \rangle$ for all tenants (generalizing PS-P [58]), and $\mathcal{M}_k = \frac{1}{M}$ for flows on a single link (generalizing per-flow max-min fairness [43]).
Isolation guarantee is defined as the lowest progress across all tenants, i.e., $\min_k \mathcal{M}_k$.

2. **High Utilization:** Spare network capacities should be utilized by tenants with elastic demands to ensure high utilization as long as it does not decrease anyone's progress.
A related concept is *work conservation*, which ensures that either a link is fully utilized or demands from all flows traversing the link have been satisfied [43, 58]. Although existing research conflates the two [14, 15, 45, 51, 58, 59, 61, 62, 67], we show in the next section why that is not the case.
3. **Proportionality:** A tenant's bandwidth allocation should be proportional to its payment similar to resources like CPU and memory. We discuss this requirement in more details in Section 3.3.1.

2.3 Challenges and Inefficiencies of Existing Allocation Policies

Prior work also identified two tradeoffs: *isolation guarantee vs. proportionality* and *high utilization vs. proportionality*. However, it has been implicitly assumed that tenant-level *optimal* isolation guarantee[5] and network-level work conservation can coexist. Although optimal isolation guarantee and network-level work conservation can coexist for a single link – max-min fairness is an example – *optimal isolation guarantee and work conservation can be at odds when we consider the network as a whole.* This has several implications on both isolation guarantee and network utilization. In particular, we can (1) either optimize utilization, *then* maximize the isolation guarantee with best effort; or (2) optimize the isolation guarantee, *then* maximize utilization with best effort.[6] Please refer to Appendix C for more details.

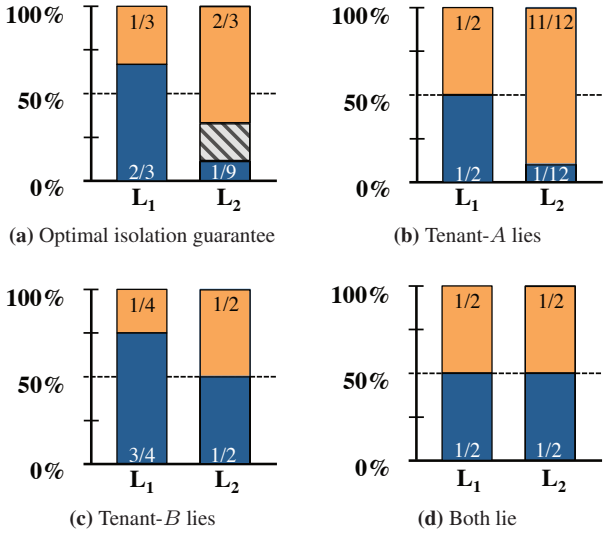


Figure 5: Bandwidth consumptions of tenant-A (orange) and tenant-B (dark blue) with correlation vectors $\overrightarrow{d_A} = \langle \frac{1}{2}, 1 \rangle$ and $\overrightarrow{d_B} = \langle 1, \frac{1}{6} \rangle$, when both run elastic-demand applications. (a) Optimal isolation guarantee in the absence of work conservation. With work conservation, (b) tenant-A increases her progress at the expense of tenant-B, and (c) tenant-B can do the same, which results in (d) a prisoner's dilemma.

2.3.1 Full Utilization but Suboptimal Isolation Guarantee

As shown in prior work [58, Section 2.5], flow-level and VM-level mechanisms – e.g., per-flow, per source-destination pair [58], and per-endpoint fairness [61, 62] – can easily be manipulated by creating more flows or by using denser communication patterns. To avoid such manipulations, many allocation mechanisms [45, 58, 59] equally divide link capacities at the tenant level and allow work conservation for tenants with unmet demands. Figure 4 shows an allocation using PS-P [58] with isolation guarantee $\frac{1}{2}$. If both tenants have elastic-demand applications, they will consume entire allocations; i.e., $\overrightarrow{c_A} = \overrightarrow{c_B} = \overrightarrow{a_A} = \overrightarrow{a_B} = \langle \frac{1}{2}, \frac{1}{2} \rangle$, where $\overrightarrow{c_k} = \langle c_k^1, c_k^2, \ldots c_k^{2P} \rangle$ and c_k^i is the fraction of link-i *consumed* by tenant-k. Recall that a_k^i is the guaranteed allocation of link-i to tenant-k.

However, PS-P and similar mechanisms are also suboptimal. For the ongoing example, Figure 5a shows the optimal isolation guarantee of $\frac{2}{3}$, which is higher than that provided by PS-P. In short, full utilization does not necessarily imply optimal isolation guarantee!

[4] We are continuing the example in Figure 3 but omitted the rest of $\overrightarrow{a_k}$, because there is either no contention or they are symmetric.

[5] Optimality means that the allocation maximizes the isolation guarantee across all tenants, i.e., **maximize** $\left\{ \min_k \mathcal{M}_k \right\}$.

[6] Maximizing a combination of these two is also an interesting future direction.

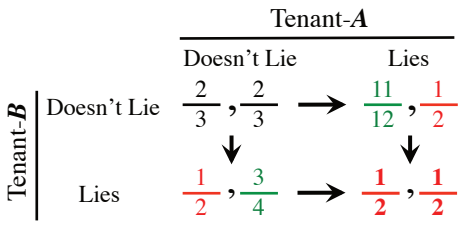


Figure 6: Payoff matrix for the example in Section 2.3. Each cell shows progress of tenant-A and tenant-B.

2.3.2 Optimal Isolation Guarantee but Low Utilization

In contrast, optimal isolation guarantee does not necessarily mean full utilization. In general, optimal isolation guarantees can be calculated using DRF [33], which generalizes max-min fairness to multiple resources. In the example of Figure 5a, each uplink and downlink of the fabric is an independent resource – $2P$ in total.

Given this premise, it seems promising and straightforward to keep the DRF-component for optimal isolation guarantee and strategy-proofness and try to ensure full utilization by allocating *all* remaining resources.In the following two subsections, we show that work conservation may render isolation guarantee no longer optimal, and even worse, may reduce useful network utilization.

2.3.3 Naive Work Conservation Reduces Optimal Isolation Guarantee

We first illustrate that even the optimal isolation guarantee allocation degenerates into the classic prisoner's dilemma problem [30] in the presence of work conservation. In particular, we show that reporting a false correlation vector $\langle 1, 1\rangle$ is the dominant strategy for each tenant, i.e., her best option, no matter whether the other tenants tell the truth or not. As a consequence, optimal isolation guarantees decrease (Figure 6).

If tenant-A can use the spare bandwidth in link-2, she can increase her progress at the expense of tenant-B by changing her correlation vector to $\overrightarrow{d'_A} = \langle 1, 1\rangle$. With an unmodified $\overrightarrow{d_B} = \langle 1, \frac{1}{6}\rangle$, the new allocation would be $\overrightarrow{a_A} = \langle \frac{1}{2}, \frac{1}{2}\rangle$ and $\overrightarrow{a_B} = \langle \frac{1}{2}, \frac{1}{12}\rangle$. However, work conservation would increase it to $\overrightarrow{a_A} = \langle \frac{1}{2}, \frac{11}{12}\rangle$ (Figure 5b). Overall, progress of tenant-A would increase to $\frac{11}{12}$, while decreasing it to $\frac{1}{2}$ for tenant-B. As a result, the isolation guarantee decreases from $\frac{2}{3}$ to $\frac{1}{2}$.

The same is true for tenant-B as well. Consider again that only tenant-B reports a falsified correlation vector $\overrightarrow{d'_B} = \langle 1, 1\rangle$ to receive a favorable allocation: $\overrightarrow{a_A} = \langle \frac{1}{4}, \frac{1}{2}\rangle$ and $\overrightarrow{a_B} = \langle \frac{1}{2}, \frac{1}{2}\rangle$. Work conservation would increase it to $\overrightarrow{a_B} = \langle \frac{3}{4}, \frac{1}{2}\rangle$ (Figure 5c). Overall, progress of tenant-B would increase to $\frac{3}{4}$, while decreasing it to $\frac{1}{2}$ for tenant-A, resulting in the same suboptimal isolation guarantee $\frac{1}{2}$.

Since both tenants gain by lying, they would both simultaneously lie: $\overrightarrow{d'_A} = \overrightarrow{d'_B} = \langle 1, 1\rangle$, resulting in a lower isolation guarantee $\frac{1}{2}$ (Figure 5d). Both are worse off!

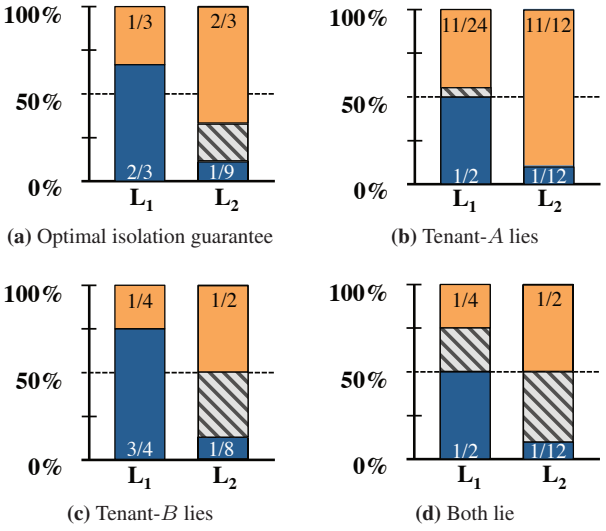


Figure 7: Bandwidth consumptions of tenant-A (orange) and tenant-B (dark blue) with correlation vectors $\overrightarrow{d_A} = \langle \frac{1}{2}, 1\rangle$ and $\overrightarrow{d_B} = \langle 1, \frac{1}{6}\rangle$, when neither runs elastic-demand applications. (a) Optimal isolation guarantee allocation is not work-conserving. With work conservation, (b) utilization can increase or (c) decrease, based on which tenant lies. (d) However, ultimately it lowers utilization. Shaded means unallocated.

In this example, the inefficiency arises due to allocating all spare resources to the tenant who demands more. We show in Appendix B that intuitive allocation policies of all spare resources – e.g., allocating all to who demands the least, allocating equally to all tenants with non-zero demands, and allocating proportionally to tenants' demands – do not work as well.

2.3.4 Naive Work Conservation can Even Decrease Utilization

Now consider that *neither* tenant has elastic-demand applications; i.e., they can only consume bandwidth proportional to their correlation vectors. A similar prisoner's dilemma unfolds (Figure 6), but this time, network utilization decreases as well.

Given the optimal isolation guarantee allocation, $\overrightarrow{a_A} = \overrightarrow{c_A} = \langle \frac{1}{3}, \frac{2}{3}\rangle$ and $\overrightarrow{a_B} = \overrightarrow{c_B} = \langle \frac{2}{3}, \frac{1}{9}\rangle$, both tenants have the same optimal isolation guarantee: $\frac{2}{3}$, and $\frac{2}{9}$-th of link-2 remain unused (Figure 7a). One would expect work conservation to utilize this spare capacity.

Same as before, if tenant-A changes her correlation vector to $d'_A = \langle 1, 1\rangle$, she can receive an allocation $\overrightarrow{a_A} = \langle \frac{1}{2}, \frac{11}{12}\rangle$ and consume $\overrightarrow{c_A} = \langle \frac{11}{24}, \frac{11}{12}\rangle$. This increases her isolation guarantee to $\frac{11}{12}$ and total network utilization increases (Figure 7b).

Similarly, tenant-B can receive an allocation $\overrightarrow{a_B} = \langle \frac{3}{4}, \frac{1}{2}\rangle$ and consume $\overrightarrow{c_B} = \langle \frac{3}{4}, \frac{1}{8}\rangle$ to increase her isola-

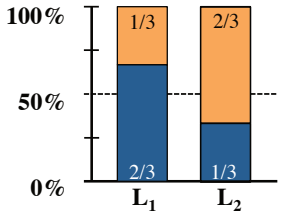


Figure 8: Optimal allocations with maximum achievable utilizations and maximum isolation guarantees for tenant-A (orange) and tenant-B (dark blue).

tion guarantee to $\frac{3}{4}$. Utilization decreases (Figure 7c).

Consequently, both tenants lie and consume $\overrightarrow{c_A} = \langle \frac{1}{4}, \frac{1}{2} \rangle$ and $\overrightarrow{c_B} = \langle \frac{1}{2}, \frac{1}{12} \rangle$ (Figure 7d). Instead of increasing, work conservation decreases network utilization!

2.4 Summary

The primary takeaways of this section are the following:

- Existing mechanisms provide either suboptimal isolation guarantees or low network utilization.
- There exists a strong tradeoff between optimal isolation guarantee and high utilization in a multi-tenant network. The key lies in strategy-proofness: optimal isolation guarantee requires it, while work conservation nullifies it. We provide a formal result about this (**Corollary 2**) in the next section.
- Unlike single links, work conservation can decrease network utilization instead of increasing it.

3 HUG: Analytical Results

In this section, we show that despite the tradeoff between optimal isolation guarantee and work conservation, it is possible to increase utilization to some extent. Moreover, we present HUG, the optimal algorithm to ensure maximum achievable utilization *without* sacrificing optimal isolation guarantees and strategy-proofness of DRF.

We defer the proofs from this section to Appendix A.

3.1 Root Cause Behind the Tradeoff: Unrestricted Sharing of Spare Resources

Going back to Figure 5, both tenants were incentivized to lie because they were receiving spare resources without any restriction due to the pursuit of work conservation.

After tenant-A lied in Figure 5b, both $\mathcal{M}_A$ and $\mathcal{M}_B$ decreased to $\frac{1}{2}$. However, by cheating, tenant-A managed to increase her allocation in link-1 to $\frac{1}{2}$ from $\frac{1}{3}$. Next, indiscriminate work conservation increased her allocation in link-2 to $\frac{11}{12}$ from the initial $\frac{1}{2}$, effectively increasing $\mathcal{M}_A$ to $\frac{11}{12}$. Similarly in Figure 5c, tenant-B first increased her allocation in link-2 to $\frac{1}{2}$ from $\frac{1}{9}$ and then work conservation increased her allocation in link-1 to $\frac{3}{4}$ from the initial $\frac{1}{2}$.

Algorithm 1 High Utilization with Guarantees (HUG)

Input: $\{\overrightarrow{d_k}\}$: *reported* correlation vector of tenant-k, $\forall k$
Output: $\{\overrightarrow{a_k}\}$: guaranteed resource allocated to tenant-k, $\forall k$

Stage 1: Calculate the optimal isolation guarantee ($\mathcal{M}^*$) and minimum allocations $\overrightarrow{a_k} = \mathcal{M}^* \overrightarrow{d_k},\ \forall k$

Stage 2: Restrict maximum utilization for each of the $2P$ links, such that $c_k^i \leq \mathcal{M}^*,\ \forall i, \forall k$

Consequently, we must eliminate a tenant's incentive to gain too much spare resources by lying; i.e., *a tenant should never be able to manipulate and increase her progress due to work conservation.*

Lemma 1 *Any allocation policy with the following two characteristics is not strategyproof:*

1. *it first uses DRF to ensure the optimal isolation guarantee and then assigns the spare, DRF-unallocated resources for work conservation;*
2. *there exists at least one tenant whose allocation (including spare) on some link is more than her progress under DRF based on her reported correlation vector.*

Corollary 2 (of Lemma 1) *Optimal isolation guarantee allocations cannot always be work-conserving even in the presence of elastic-demand applications.* □

3.2 The Optimal Algorithm: HUG

Given the tradeoff, our goal is to design an allocation algorithm that can achieve the highest utilization while keeping the optimal isolation guarantee and strategy-proofness. Formally, we want to design an algorithm to

$$\begin{aligned} \text{Maximize} \quad & \sum_{i \in [1,2P]} \sum_{k \in [1,M]} c_k^i \\ \text{subject to} \quad & \min_{k \in [1,M]} \mathcal{M}_k = \mathcal{M}^*, \end{aligned} \tag{1}$$

where c_k^i is the actual consumption[7] of tenant-k on link-i for allocation a_k^i, and $\mathcal{M}^*$ is the optimal isolation guarantee.

We observe that an optimal algorithm would have restricted tenant-A's progress in Figure 5b and tenant-B's progress in Figure 5c to $\frac{2}{3}$. Consequently, they would not have been incentivized to lie and the prisoner's dilemma could have been avoided. Algorithm 1 – referred to as *High Utilization with Guarantees* (HUG) – is such a two-stage allocation mechanism that guarantees maximum utilization while maximizing the isolation guarantees across tenants and is strategyproof.

In the first stage, HUG allocates resources to maximize isolation guarantees across tenants. To achieve this, we pose our problem as a $2P$-resource fair sharing problem and use DRF [33, 56] to calculate $\mathcal{M}^*$. By reserving

[7] Consumptions can differ from allocations when tenants are lying.

these allocations, HUG ensures isolation. Moreover, because DRF is strategy-proof, tenants are guaranteed to use these allocations (i.e., $c_k^i \geq a_k^i$).

While DRF maximizes the isolation guarantees (a.k.a. dominant shares), it results in low network utilization. In some cases, *DRF may even have utilization arbitrarily close to zero, and HUG can increase that to 100%* (**Lemma 6**).

To achieve this, the second stage of HUG maximizes utilization while still keeping the allocation strategyproof. In this stage, we calculate upper bounds to restrict how much of the spare capacity a tenant can use in each link, with the constraint that the largest share across all links cannot increase (**Lemma 1**). As a result, Algorithm 1 remains strategy-proofness across both stages. Because spare usage restrictions can be applied locally, HUG can be enforced in individual machines.

Illustrated in Figure 8, the bound is set at $\frac{2}{3}$ for both tenants, and tenant-B can use its elastic demand on link-2's spare resource, while tenant-A cannot as she has reached its bound on link-2.

3.3 HUG Properties

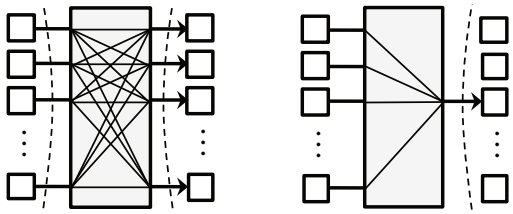
We list the main properties of HUG in the following.

1. In non-cooperative cloud environments, HUG is strategyproof (**Theorem 3**), maximizes isolation guarantees (**Corollary 4**), and ensures the highest utilization possible for an optimal isolation guarantee allocation (**Theorem 5**). In particular, **Lemma 6** shows that under some cases, DRF may have utilization arbitrarily close to 0, and HUG improves it to 100%. We defer the proofs of properties in the Section to Appendix A.
2. In cooperative environments like private datacenters, HUG maximizes isolation guarantees and is work-conserving. Work conservation is achievable because strategy-proofness is a non-requirement in this case.
3. Because HUG provides optimal isolation guarantee, it provides min-cut proportionality (§ 3.3.1) in both non-cooperative and cooperative environments.

Regardless of resource types, the identified tradeoff exists in general multi-resource allocation problems and HUG can directly be applied.

3.3.1 Min-Cut Proportionality

Prior work promoted the notion of *proportionality* [58], where tenants would expect to receive total allocations proportional to their number of VMs *regardless* of communication patterns. Meaning, two tenants, each with N VMs, should receive equal bandwidth even if tenant-X has an all-to-all communication pattern (i.e., $\overrightarrow{d_X} = \langle 1, 1, \ldots, 1 \rangle$) and tenant-Y has an N-to-1 pattern (i.e., exactly one 1 in $\overrightarrow{d_Y}$ and the rest are zeros). Figure 9



(a) Tenant-X (N-to-N) **(b)** Tenant-Y (N-to-1)

Figure 9: Communication patterns of tenant-X and tenant-Y with (a) two minimum cuts of size P, where P is the number of fabric ports, and (b) one minimum cut of size 1. The size of the minimum cut of a communication pattern determines its effective bandwidth even if it were placed alone.

shows an example. Clearly, tenant-Y will be bottlenecked at her only receiver; trying to equalize them will only result in low utilization. As expected, FairCloud proved that such proportionality is not achievable as it decreases both isolation guarantee and utilization [58]. None of the existing algorithms provide proportionality.

Instead, we consider a relaxed notion of proportionality, called *min-cut proportionality*, that depends on communication patterns and ties proportionality with a tenant's progress. Specifically, each tenant receives minimum bandwidth proportional to the size of the *minimum cut* [31] of their communication patterns. Meaning, in the earlier example, tenant-X would receive P times more total bandwidth than tenant-Y, but they would have the optimal isolation guarantee ($\mathcal{M}_X = \mathcal{M}_Y = \frac{1}{2}$).

Min-cut proportionality and optimal isolation guarantee can coexist, but they both have tradeoffs with work conservation.

4 Design Details

This section discusses how a cloud operator can implement, enforce, and expose HUG to the tenants (§4.1), how to exploit placement to further improve HUG's performance (§4.2), and how HUG can handle weighted, heterogeneous scenarios (§4.3).

4.1 Architectural Overview

HUG can easily be implemented atop existing monitoring infrastructure of cloud operators (e.g., Amazon CloudWatch [1]). Tenants would periodically update their correlation vectors through a public API, and the operator would compute new allocations and update enforcing agents within milliseconds.

HUG API The tenant-facing API simply transfers a tenant's correlation vector ($\overrightarrow{d_k}$) to the operator. $\overrightarrow{d_k} = \langle 1, 1, \ldots, 1 \rangle$ is used as the default correlation vector. By design, HUG incentivizes tenants to report and maintain accurate correlation vectors. This is because the more accurate it is – instead of the default $\overrightarrow{d_k} = \langle 1, 1, \ldots, 1 \rangle$ –

the higher are her progress and performance.

Many applications already know their long-term profiles (e.g., multi-tier online services [19, 52]) and others can calculate on the fly (e.g., bulk communication data-intensive applications [22, 23]). Moreover, existing techniques in traffic engineering can provide good accuracy in estimating and predicting demand matrices for coarse time granularities [17, 18, 20, 47, 48].

Centralized Computation For any update, the operator must run Algorithm 1. Although Stage-1 requires solving a linear program to determine the optimal isolation guarantee (i.e., the DRF allocation) [33], it can also be rewritten as a closed-form equation [56] when tenants can scale up and down following their normalized correlation vectors. The progress of all tenants after Stage-1 of Algorithm 1 – the optimal isolation guarantee – is:

$$\mathcal{M}^* = \frac{1}{\max\limits_{1 \le i \le 2P} \sum\limits_{k=1}^{M} d_k^i} \tag{2}$$

Equation (2) is computationally inexpensive. For our 100-machine cluster, calculating $\mathcal{M}^*$ takes about 5 microseconds. Communicating the decision to all 100 machines takes just 8 milliseconds and to 100,000 (emulated) machines takes less than 1 second (§5.1.2).

The guaranteed minimum allocations of tenant-k can then be calculated as $a_k^i = \mathcal{M}^* d_k^i$ for all $1 \le i \le 2P$.

Local Enforcement Enforcement in Stage-2 of Algorithm 1 is simple as well. After reserving the minimum uplink and downlink allocations for each tenant, each machine needs to ensure that no tenant can consume more than $\mathcal{M}^*$ fraction of the machine's up or down link capacities (C^i) to the network; i.e., $a_k^i \le c_k^i \le \mathcal{M}^*$. The spare is allocated among tenants using local max-min fairness *subject to* tenant-specific upper-bounds.

Because we only care about inter-tenant behavior – not how a tenant performs internal sharing – stock Linux `tc` is sufficient (§5). A tenant has the flexibility to choose from traditional per-flow fairness, shortest-first flow scheduling [12, 41], or explicit rate-based flow control [29].

4.2 VM Placement and Re-Placement/Migration

While $\mathcal{M}^*$ is optimal for a *given* placement, it can be improved by changing the placement of tenant VMs based on their correlation vectors. One must perform load balancing across all machines to minimize the denominator of Equation (2). Cloud operators can employ optimization frameworks like [19] to perform initial VM placement and periodic migrations with an additional load balancing constraint. However, VM placement is a notoriously difficult problem because of often-incompatible constraints like fault-tolerance and collocation [19], and we consider its detailed study an important future work. It is worth noting that with *any* VM placement, HUG provides the highest attainable utilization without sacrificing optimal isolation guarantee and strategy-proofness.

4.3 Additional Constraints

Weighted Tenants Giving preferential treatment to tenants is simple. Just using $w_k \overrightarrow{d_k}$ instead of $\overrightarrow{d_k}$ in Equation (2) would account for tenant weights (w_k for tenant-k) in calculating $\mathcal{M}^*$.

Heterogeneous Capacities Because allocations are calculated independently in each machine based on $\mathcal{M}^*$ and local capacities (C^i), HUG supports heterogeneous link capacities.

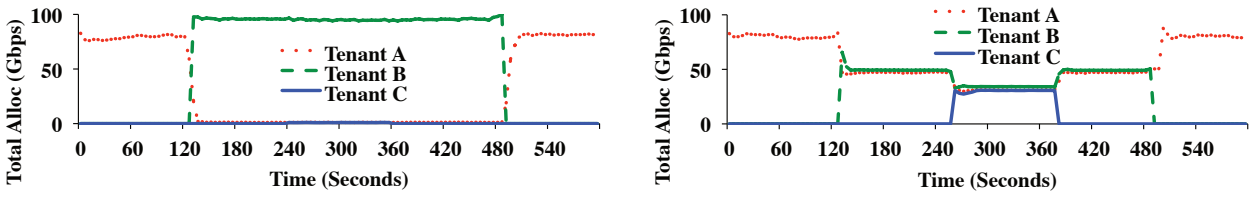
Bounded Demands So far we have considered only elastic tenants. If tenant-k has bounded demands, i.e., $d_k^i < 1$ for all $i \in [1, 2P]$, calculating a common $\mathcal{M}^*$ and corresponding $\overrightarrow{a_k}$ in *one round* using Equation (2) will be inefficient. This is because tenant-k might require less than the calculated allocation, and being bounded, she cannot elastically scale up to use it. Instead, we must use the *multi-round* DRF algorithm [56, Algorithm 1] in Stage-1 of HUG; Stage-2 will remain the same. Note that this is similar to max-min fairness in a single link when a flow has a smaller demand than its $\frac{1}{n}$-th share.

5 Evaluation

We evaluated HUG using trace-driven simulations and EC2 deployments. Our results show the following:

- HUG isolates multiple tenants across the entire network, and it can scale up to 100,000 machines with less than one second overhead (§5.1).
- HUG ensures the optimal isolation guarantee – almost 7000× more than per-flow fairness and about 7.4× more than PS-P in production traces – while providing 1.4× higher utilization than DRF (§5.2).
- HUG outperforms per-flow fairness (PS-P) by 17.35× (1.48×) in terms of the 95th percentile slowdown and by 1.49× (1.14×) in minimizing the average shuffle completion time (§5.3).
- HUG outperforms Varys [23] in terms of the maximum shuffle completion time by 1.77×, even though Varys is 1.45× better in minimizing the average shuffle completion time and 1.33× better in terms of the 95th percentile slowdown (§5.3).

We present our results in three parts. First, we microbenchmark HUG on 100-machine EC2 clusters to evaluate HUG's guarantees and overheads (§5.1). Second, we leverage traces collected from a 3200-machine Facebook cluster by Popa et al. [58] to compare HUG's *instantaneous* allocation characteristics with that of per-



(a) Per-flow Fairness (TCP) **(b)** HUG

Figure 10: [EC2] Bandwidth consumptions of three tenants arriving over time in a 100-machine EC2 cluster. Each tenant has 100 VMs, but each uses a different communication pattern (§5.1.1). We observe that (a) using TCP, tenant-B dominates the network by creating more flows; (b) HUG isolates tenants A and C from tenant B.

flow fairness, PS-P [58], and DRF [33] (§5.2). Finally, we evaluate HUG's *long-term* impact on application performance using a 3000-machine Facebook cluster trace used by Chowdhury et al. [23] and compare against per-flow fairness, PS-P, DRF, as well as Varys, which focuses only on improving performance (§5.3).

5.1 Testbed Experiments

Methodology We performed our experiments on 100 `m2.4xlarge` Amazon EC2 [2] instances running on Linux kernel 3.4.37 and used the default `htb` and `tc` implementations. While there exist proposals for more accurate `qdisc` implementations [45, 57], the default `htb` worked sufficiently well for our purposes. Each of the machines had 1 Gbps NICs, and we could use close to full 100 Gbps bandwidth simultaneously.

5.1.1 Network-Wide Isolation

We consider a cluster with 100 EC2 machines, divided between three tenants A, B, and C that arrive over time. Each tenant has 100 VMs; i.e., VMs A_i, B_i, and C_i are collocated on the i-th physical machine. However, they have different communication patterns: tenants A and C have pairwise one-to-one communication patterns (100 VM-VM flows each), whereas tenant-B follows an all-to-all pattern using $10,000$ flows. Specifically, A_i communicates with $A_{(i+50)\%100}$, C_j communicates with $C_{(j+25)\%100}$, and any B_k communicates with all B_l, where $i, j, k, l \in \{1, ..., 100\}$. Each tenant demands the entire capacity at each machine; hence, the entire capacity of the cluster should be equally divided among the active tenants to maximize isolation guarantees.

Figure 10a shows that as soon as tenant-B arrives, she takes up the entire capacity in the absence of isolation guarantee. Tenant-C receives only marginal share as she arrives after tenant-B and leaves before her. Note that tenant-A (when alone) uses only about 80% of the available capacity; this is simply because just one TCP flow per VM-VM pair often cannot saturate the link.

Figure 10b presents the allocation using HUG. As tenants arrive and depart, allocations are dynamically calculated, propagated, and enforced in each machine of the cluster. As before, tenants A and C use marginally less than their allocations because of creating only one flow between each VM-VM pair.

5.1.2 Scalability

The key challenge in scaling HUG is its centralized resource allocator, which must recalculate tenant shares and redistribute them across the entire cluster whenever any tenant changes her correlation vector.

We found that the time to calculate new allocations using HUG is less than 5 microseconds in our 100 machine cluster. Furthermore, a recomputation due to a tenant's arrival, departure, or change of correlation vector would take about 8.6 milliseconds on average for a $100,000$-machine datacenter.

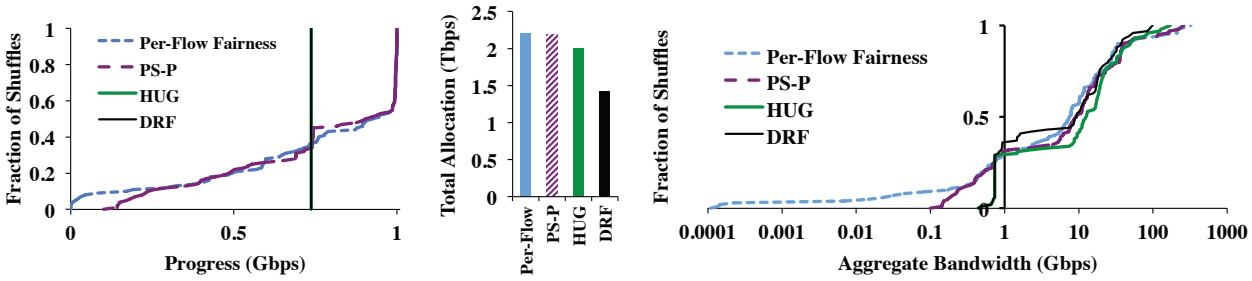
Communicating a new allocation takes less than 10 milliseconds to 100 machines and around 1 second for $100,000$ *emulated* machines (i.e., sending the same message 1000 times to each of the 100 machines).

5.2 Instantaneous Fairness

While Section 5.1 evaluated HUG in controlled, synthetic scenarios, this section focuses on HUG's instantaneous allocation characteristics in the context of a large-scale cluster.

Methodology We use a one-hour snapshot with 100 concurrent jobs from a production MapReduce trace, which was extracted from a 3200-machine Facebook cluster by Popa et al. [58, Section 5.3]. Machines are connected to the network using 1 Gbps NICs. In the trace, a job with M mappers and R reducers – hence, the corresponding $M \times R$ shuffle – is described as a matrix with the amount of data to transfer between each M-R pair. We calculated the correlation vectors of individual shuffles from their communication matrices ourselves using the optimal rate allocation algorithm for a single shuffle [22, 23], ensuring *all* the flows of each shuffle to finish simultaneously.

Given the workload, we calculate progress of each job/shuffle using different allocation mechanisms and



(a) Distribution of progress **(b)** Total allocation **(c)** Distribution of allocations

Figure 11: [Simulation] HUG ensures higher isolation guarantee than high-utilization schemes like per-flow fairness and PS-P, and provides higher utilization than multi-resource fairness schemes like DRF.

cross-examine characteristics like isolation guarantee, utilization, and proportionality.

5.2.1 Impact on Progress

Figure 11a presents the distribution of progress of each shuffle. Recall that the progress of a shuffle – we consider each shuffle an individual tenant in this section – is the amount of bandwidth it is receiving in its bottleneck up or downlink (i.e., progress can be at most 1 Gbps). Both HUG and DRF (overlapping vertical lines in Figure 11a) ensure the same progress (0.74 Gbps) for all shuffles. Note that despite same progress, shuffles will finish at different times based on how much data each one has to send (§5.3). Per-flow fairness and PS-P provide very wide ranges: 112 Kbps to 1 Gbps for the former and 0.1 Gbps to 1 Gbps for the latter. Shuffles with many flows crowd out the ones with fewer flows under per-flow fairness, and PS-P suffers by ignoring correlation vectors and through indiscriminate work conservation.

5.2.2 Impact on Utilization

By favoring heavy tenants, per-flow fairness and PS-P do succeed in their goals of increasing network utilization (Figure 11b). Given the communication patterns of the workload, the former utilizes 69% of 3.2 Tbps total capacity across all machines and the latter utilizes 68.6%. In contrast, DRF utilizes only 45%. HUG provides a common ground by extending utilization to 62.4% without breaking strategy-proofness and providing optimal isolation guarantee.

Figure 11c breaks down total allocations of each shuffle and demonstrates two high-level points:

1. HUG ensures overall higher utilization (1.4× on average) than DRF by ensuring equal progress for smaller shuffles and by using up additional bandwidth for larger shuffles. It does so while ensuring the same optimal isolation guarantee as DRF.
2. Per-flow fairness crosses HUG at the 90-th percentile; i.e., the top 10% shuffles receive more bandwidth than they do under HUG, while the other 90% receive less than they do using HUG. PS-P crosses over at the 76-th percentile.

Bin	1 (SN)	2 (LN)	3 (SW)	4 (LW)
% of Shuffles	52%	16%	15%	17%

Table 2: Shuffles binned by their lengths (**S**hort and **L**ong) and widths (**N**arrow and **W**ide).

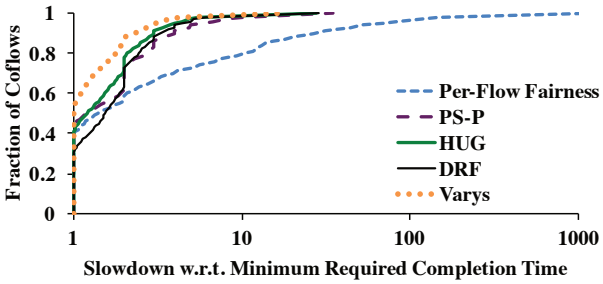
5.2.3 Impact on Proportionality

A collateral benefit of HUG is that tenants receive allocations proportional to their bottleneck demands. Consequently, despite the same progress across all shuffles (Figure 11a), their total allocations vary (Figure 11c) based on the size of minimum cuts in their communication patterns.

5.3 Long-Term Characteristics

We have shown in Section 5.2 that HUG provides optimal isolation guarantee in the *instantaneous* case. However, similar to all instantaneous solutions [33, 43, 58], HUG does not provide any *long-term* isolation or fairness guarantees. Consequently, in this section, we evaluate HUG's long-term impact on performance using a production trace through simulations.

Methodology For these simulations, we use a MapReduce/Hive trace from a 3000-machine production Facebook cluster. The trace includes the arrival times, communication matrices, and placements of tasks of over 10,000 shuffle during one day. Shuffles in this trace have diverse length (i.e., size of the longest flow) and width (i.e., the number of flows) characteristics and roughly follow the same distribution of the original trace (Table 2). We consider a shuffle to be *short* if its longest flow is less than 5 MB and *narrow* if it has at most 50 flows; we use the same categorization. We calculated the correlation vector of each shuffle as we did before (§ 5.2).



(a) CDFs

	Min	95th	AVG	STDEV
Per-Flow Fairness	1	69.4	15.52	65.54
PS-P	1	5.9	2.22	2.97
HUG	1	4.0	1.86	2.25
DRF	1	4.4	2.11	2.66
Varys	1	3.0	1.43	0.99

(b) Summaries

Figure 12: [Simulation] Slowdowns of shuffles using different mechanisms w.r.t. the minimum completion times of each shuffle. The X-axis of (a) is in log scale.

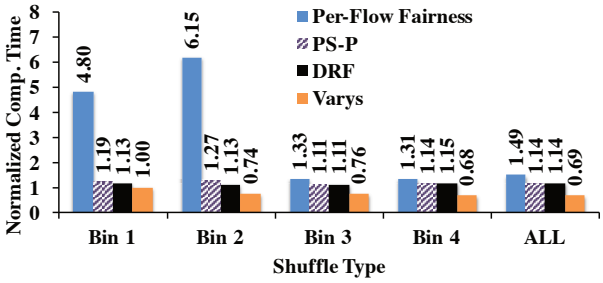


Figure 13: [Simulation] Average shuffle completion times normalized by HUG's average completion time. 95-th percentile plots are similar.

Metrics We consider two metrics: *95th percentile slowdown* and *average shuffle completion time* to respectively measure long-term progress and performance characteristics.

We define the slowdown of a shuffle as its completion time due to a scheme normalized by its minimum completion time if it were running alone; i.e.,

$$\text{Slowdown} = \frac{\text{Compared Duration}}{\text{Minimum Duration}}$$

The minimum value of slowdown is one.

We measure performance as the shuffle completion time of a scheme normalized by that using HUG; i.e.,

$$\text{Normalized Comp. Time} = \frac{\text{Compared Duration}}{\text{HUG's Duration}}$$

If the normalized completion time of a scheme is greater (smaller) than one, HUG is faster (slower).

5.3.1 Improvements Over Per-Flow Fairness

HUG improves over per-flow fairness both in terms of slowdown and performance. The 95th percentile slowdown using HUG is 17.35× better than that of per-flow fairness (Table 12b). Overall, HUG provides better slowdown across the board (Figure 12a) – 61% shuffles are better off using HUG and the rest remain the same.

HUG improves the average completion time of shuffles by 1.49× (Figure 13). The biggest wins comes from bin-1 (4.8×) and bin-2 (6.15×) that include the so-called narrow shuffles with less than 50 flows. This reinforces the fact that HUG isolates tenants with fewer flows from those with many flows. Overall, HUG performs well across all bins.

5.3.2 Improvements Over PS-P

HUG improves over PS-P in terms of the 95th percentile slowdown by 1.48×, and 45% shuffles are better off using HUG. HUG also providers better average shuffle completion times than PS-P for an overall improvement of 1.14×. Large improvements again come in bin-1 (1.19×) and bin-2 (1.27×) because PS-P also favors tenants with more flows.

Note that instantaneous high utilization of per-flow fairness and PS-P (§5.2) does not help in the long run due to lower isolation guarantee.

5.3.3 Improvements Over DRF

While HUG and DRF has the same worst-case slowdown, 70% shuffles are better off using HUG. HUG also provides better average shuffle completion times than DRF for an overall improvement of 1.14×.

5.3.4 Comparison to Varys

Varys outperforms HUG by 1.33× in terms of the 95th percentile slowdown and by 1.45× in terms of average shuffle completion time. However, because Varys attempts to improve the average completion time by prioritization, it risks in terms of the maximum completion time. More precisely, HUG outperforms Varys by 1.77× in terms of the maximum shuffle completion time (not shown).

6 Discussion

Payment Model Similar to many existing proposals [32, 33, 45, 46, 58, 59, 61, 62], we assume that tenants pay per-hour flat rates for individual VMs, but there is no pricing model associated with their network usage. This is also the prevalent model of resource pricing in cloud computing [2, 5, 8]. Exploring *whether* and *how* a network pricing model would change our solution and *what* that model would look like requires further attention.

Determining Correlation Vectors Unlike long-term correlation vectors, e.g., over the course of an hour or for

an entire shuffle, accurately capturing short-term changes can be difficult. How fast tenants should update their vectors and whether that is faster than centralized HUG can react to requires additional analysis.

Decentralized HUG HUG's centralized design makes it easier to analyze its properties and simplifies its implementation. We believe that designing a decentralized version of HUG is an important future work, which will be especially relevant for sharing wide-area networks in the context of geo-distributed analytics [60, 66].

7 Related Work

Single-Resource Fairness Max-min fairness was first proposed by Jaffe [43] to ensure at least $\frac{1}{n}$-th of a link's capacity to each flow. Thereafter, many mechanisms have been proposed to achieve it, including weighted fair queueing (WFQ) [25, 55] and those similar to or extending WFQ [16, 34, 35, 63, 64]. We generalize max-min fairness to parallel communication observed in scale-out applications, showing that unlike in the single-link scenario, optimal isolation guarantee, strategy-proofness, and work conservation cannot coexist.

Multi-Resource Fairness Dominant Resource Fairness (DRF) [33] maximizes the dominant share of each user in a strategyproof manner. Solutions that have attempted to improve the system-level efficiency of multi-resource allocation – both before [54, 65] and after [27, 38, 56] DRF – sacrifice strategy-proofness. We have proven that work-conserving allocation without strategy-proofness can hurt utilization instead of improving it.

Dominant Resource Fair Queueing (DRFQ) [32] approximates DRF over time in *individual* middleboxes. In contrast, HUG generalizes DRF to environments with elastic demands to increase utilization across the *entire* network and focuses only on instantaneous fairness.

Joe-Wong et al. [46] have presented a unifying framework to capture fairness-efficiency tradeoffs in multi-resource environments. They assume a *cooperative* environment, where tenants never lie. HUG falls under their FDS family of mechanisms. In non-cooperative environments, however, we have shown that the interplay between work conservation and strategy-proofness is critical, and our work complements the framework of [46].

Network-Wide / Tenant-Level Fairness Proposals for sharing cloud networks range from static allocation [13, 14, 44] and VM-level guarantees [61, 62] to variations of network-wide sharing mechanisms [45, 51, 58, 59, 67]. We refer the reader to the survey by Mogul and Popa [53] for an overview. FairCloud [58] stands out by systematically discussing the tradeoffs and addresses several limitations of other approaches. Our work generalizes FairCloud [58] and many proposals similar to FairCloud's PS-P policy [45, 59, 61]. When all tenants have elastic demands, i.e., all correlation vectors have all elements as 1, we give the same allocation; for all other cases, we provide higher isolation guarantee and utilization.

Efficient Schedulers Researchers have also focused on efficient scheduling and/or packing of datacenter resources to minimize job and communication completion times [12, 21–23, 26, 36, 41]. Our work is orthogonal and complementary to these work focusing on application-level efficiency within each tenant. We guarantee isolation across tenants, so that each tenant can internally perform whatever efficiency or fairness optimizations among her own applications.

8 Conclusion

In this paper, we have proved that there is a strong tradeoff between optimal isolation guarantees and high utilization in non-cooperative public clouds. We have also proved that work conservation can decrease utilization instead of improving it, because no network sharing algorithm remains strategyproof in its presence.

To this end, we have proposed HUG to restrict bandwidth utilization of each tenant to ensure highest utilization with optimal isolation guarantee across multiple tenants in non-cooperative environments. In cooperative environments, where strategy-proofness might be a non-requirement, HUG simultaneously ensures both work conservation and the optimal isolation guarantee.

HUG generalizes single-resource max-min fairness to multi-resource environments where a tenant's demand on different resources are correlated and elastic. In particular, it provides optimal isolation guarantee, which is significantly higher than that provided by existing multi-tenant network sharing algorithms. HUG also complements DRF with provably highest utilization without sacrificing other useful properties of DRF. Regardless of resource types, the identified tradeoff exists in general multi-resource allocation problems, and all those scenarios can take advantage of HUG.

Acknowledgments

We thank our shepherd Mohammad Alizadeh and the anonymous reviewers of SIGCOMM'15 and NSDI'16 for useful feedback. This research is supported in part by NSF CISE Expeditions Award CCF-1139158, NSF Award CNS-1464388, DOE Award SN10040 DE-SC0012463, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, IBM, SAP, The Thomas and Stacey Siebel Foundation, Adatao, Adobe, Apple, Inc., Blue Goji, Bosch, Cisco, Cray, Cloudera, EMC2, Ericsson, Facebook, Fujitsu, Guavus, HP, Huawei, Informatica, Intel, Microsoft, NetApp, Pivotal, Samsung, Schlumberger, Splunk, Virdata, and VMware.

References

[1] Amazon CloudWatch. http://aws.amazon.com/cloudwatch.

[2] Amazon EC2. http://aws.amazon.com/ec2.

[3] Apache Hadoop. http://hadoop.apache.org.

[4] AWS Innovation at Scale. http://goo.gl/Py2Ueo.

[5] Google Compute Engine. https://cloud.google.com/compute.

[6] Google Container Engine. http://kubernetes.io.

[7] A look inside Google's data center networks. http://goo.gl/u0vZCY.

[8] Microsoft Azure. http://azure.microsoft.com.

[9] Storm: Distributed and fault-tolerant realtime computation. http://storm-project.net.

[10] Trident: Stateful Stream Processing on Storm. http://goo.gl/cKsvbj.

[11] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM*, 2014.

[12] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. Mckeown, B. Prabhakar, and S. Shenker. pFabric: Minimal near-optimal datacenter transport. In *SIGCOMM*, 2013.

[13] S. Angel, H. Ballani, T. Karagiannis, G. OShea, and E. Thereska. End-to-end performance isolation through virtual datacenters. In *OSDI*, 2014.

[14] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *SIGCOMM*, 2011.

[15] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. OShea. Chatty tenants and the cloud network sharing problem. In *NSDI*, 2013.

[16] J. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *INFOCOM*, 1996.

[17] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, 2010.

[18] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine grained traffic engineering for data centers. In *CoNEXT*, 2011.

[19] P. Bodik, I. Menache, M. Chowdhury, P. Mani, D. Maltz, and I. Stoica. Surviving failures in bandwidth-constrained datacenters. In *SIGCOMM*, 2012.

[20] M. Chowdhury, S. Kandula, and I. Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *SIGCOMM*, 2013.

[21] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *SIGCOMM*, 2015.

[22] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with Orchestra. In *SIGCOMM*, 2011.

[23] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with Varys. In *SIGCOMM*, 2014.

[24] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[25] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM*, 1989.

[26] F. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *SIGCOMM*, 2014.

[27] D. Dolev, D. G. Feitelson, J. Y. Halpern, R. Kupferman, and N. Linial. No justified complaints: On fair sharing of multiple resources. In *ITCS*, 2012.

[28] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive. A flexible model for resource management in virtual private networks. In *SIGCOMM*, 1999.

[29] N. Dukkipati. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. PhD thesis, Stanford University, 2007.

[30] M. M. Flood. Some experimental games. *Management Science*, 5(1):5–26, 1958.

[31] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[32] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica. Multi-resource fair queueing for packet processing. *SIGCOMM*, 2012.

[33] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.

[34] S. J. Golestani. Network delay analysis of a class of fair queueing algorithms. *IEEE JSAC*, 13(6):1057–1070, 1995.

[35] P. Goyal, H. M. Vin, and H. Chen. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. In *SIGCOMM*, 1996.

[36] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *SIGCOMM*, 2014.

[37] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *CoNEXT*, 2010.

[38] A. Gutman and N. Nisan. Fair allocation without trade. In *AAMAS*, 2012.

[39] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In *SIGCOMM*, 2010.

[40] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *NSDI*, 2011.

[41] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. In *SIGCOMM*, 2012.

[42] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[43] J. M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, 1981.

[44] K. Jang, J. Sherry, H. Ballani, and T. Moncaster. Silo: Predictable message completion time in the cloud. In *SIGCOMM*, 2015.

[45] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg. EyeQ: Practical network performance isolation at the edge. In *NSDI*, 2013.

[46] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *INFOCOM*, 2012.

[47] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *SIGCOMM*, 2005.

[48] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of datacenter traffic: Measurements and analysis. In *IMC*, 2009.

[49] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the "One Big Switch" abstraction in Software-Defined Networks. In *CoNEXT*, 2013.

[50] K. LaCurts, J. Mogul, H. Balakrishnan, and Y. Turner. Cicada: Introducing predictive guarantees for cloud networks. In *HotCloud*, 2014.

[51] V. T. Lam, S. Radhakrishnan, A. Vahdat, G. Varghese, and R. Pan. NetShare and stochastic NetShare: Predictable bandwidth allocation for data centers. *SIGCOMM CCR*, 42(3):5–11, 2012.

[52] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma. Application-driven bandwidth guarantees in datacenters. In *SIGCOMM*, 2014.

[53] J. C. Mogul and L. Popa. What we talk about when we talk about cloud network performance. *SIGCOMM CCR*, 42(5):44–48, 2012.

[54] J. F. Nash Jr. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

[55] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM ToN*, 1(3):344–357, 1993.

[56] D. C. Parkes, A. D. Procaccia, and N. Shah. Beyond dominant resource fairness: extensions, limitations, and indivisibilities. In *EC*, 2012.

[57] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *SIGCOMM*, 2014.

[58] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. FairCloud: Sharing the network in cloud computing. In *SIGCOMM*, 2012.

[59] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing. In *SIGCOMM*, 2013.

[60] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, V. Bahl, and I. Stoica. Low latency geo-distributed data analytics. In *SIGCOMM*, 2015.

[61] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *USENIX WIOV*, 2011.

[62] A. Shieh, S. Kandula, A. Greenberg, and C. Kim. Sharing the data center network. In *NSDI*, 2011.

[63] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM*, 1995.

[64] I. Stoica, H. Zhang, and T. Ng. A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services. In *SIGCOMM*, 1997.

[65] H. R. Varian. Equity, envy, and efficiency. *Journal of economic theory*, 9(1):63–91, 1974.

[66] A. Vulimiri, C. Curino, B. Godfrey, J. Padhye, and G. Varghese. Global analytics in the face of bandwidth and regulatory constraints. In *NSDI*, 2015.

[67] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: Incorporating time-varying network reservations in data centers. In *SIGCOMM*, 2012.

[68] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

[69] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant stream computation at scale. In *SOSP*, 2013.

A Proofs from Section 3

Proof Sketch (of Lemma 1) Consider tenant-A from the example in Figure 5. Assume that instead of reporting her true correlation vector $\overrightarrow{d_A} = \langle \frac{1}{2}, 1 \rangle$, she reports $\overrightarrow{d'_A} = \langle \frac{1}{2} + \epsilon, 1 \rangle$, where $\epsilon > 0$. As a result, her allocation will change to $\overrightarrow{a'_A} = \langle \frac{1/2+\epsilon}{3/2+\epsilon}, \frac{1}{3/2+\epsilon} \rangle$. Her allocation in link-1 $\left(\frac{1/2+\epsilon}{3/2+\epsilon}\right)$ is already larger than before $\left(\frac{1}{3}\right)$. If the work conservation policy allocates the spare resource in link-2 by δ (δ may be small but a positive value), her progress will change to $\mathcal{M}'_A = \min\left(\frac{a'^1_A}{d^1_A}, \frac{a'^2_A}{d^2_A}\right) = \min\left(\frac{1+2\epsilon}{3/2+\epsilon}, \frac{1}{3/2+\epsilon}+\delta\right)$. As long as $\epsilon < \frac{3/2\delta}{2/3-\delta}$ (if $\delta \geq \frac{2}{3}$, we have no constraint on ϵ), her progress will be better than when she was telling the truth, which makes the policy not strategyproof. The operator cannot prevent this because she knows neither a tenant's true correlation vector nor ϵ, the extent of the tenant's lie. □

Theorem 3 *Algorithm 1 is strategyproof.*

Proof Sketch (of Theorem 3) Because DRF is strategyproof, the first stage of Algorithm 1 is strategyproof as well. We show that adding the second stage does not violate strategy-proofness of the combination.

Assume that link-b is a system bottleneck – the link DRF saturated to maximize isolation guarantee in the first stage. Meaning, $b = \arg\max_i \sum_{k=1}^{M} d^i_k$. We use $D^b = \sum_{k=1}^{M} d^b_k$ to denote the total demand in link-b ($D^b \geq 1$), and $\mathcal{M}^b_k = 1/D^b$ for corresponding progress for all tenant-k ($k \in \{1, ..., M\}$) when link-b is the system bottleneck. In Figure 5, $b = 1$. The following arguments hold even for multiple bottlenecks.

Any tenant-k can attempt to increase her progress ($\mathcal{M}_k$) only by lying about her correlation vector ($\overrightarrow{d_k}$). Formally, her action space consists of all possible correlation vectors. It includes increasing and/or decreasing demands of individual resources to report a different vector, $\overrightarrow{d'_k}$ and obtain a new progress, $\mathcal{M}'_k (> \mathcal{M}_k)$. Tenant-$k$ can attempt one of the two alternatives when reporting $\overrightarrow{d'_k}$: either keep link-b still the system bottleneck or change it. We show that Algorithm 1 is strategyproof in both cases; i.e., $\mathcal{M}'_k \leq \mathcal{M}_k$.

Case 1: link-b is still the system bottleneck.

Her progress cannot improve because

- if $d'^b_k \leq d^b_k$, her share on the system bottleneck will decrease in the first stage; so will her progress. There is no spare resource to allocate in link-b.

For example, if tenant-A changes $d'^1_A = \frac{1}{4}$ instead of $d^1_A = \frac{1}{2}$ in Figure 5, her allocation will decrease to $\frac{1}{5}$-th of link-1; hence, $\mathcal{M}'_A = \frac{2}{5}$ instead of $\mathcal{M}_A = \frac{2}{3}$.

- if $d'^b_k > d^b_k$, her share on the system bottleneck will increase. However, because $D'^b > D^b$ as $d'^b_k > d^b_k$, everyone's progress including her own will decrease in the first stage ($\mathcal{M}'^b_k \leq \mathcal{M}^b_k$). The second stage will ensure that her maximum consumption in any link-i $c'^i_k \leq \max_j \left\{a'^j_k\right\}$. Therefore her progress will be smaller than

that when she tells the truth ($\mathcal{M}'^{b}_{k} < \mathcal{M}^{b}_{k}$).
For example, if tenant-A changes $d'^{1}_{A} = 1$ instead of $d^{1}_{A} = \frac{1}{2}$ in Figure 5, her allocation will increase to $\frac{1}{2}$ of link-1. However, progress of both tenants will decrease: $\mathcal{M}_A = \mathcal{M}_B = \frac{1}{2}$. The second stage will restrict her usage in link-2 to $\frac{1}{2}$ as well; hence, $\mathcal{M}'_A = \frac{1}{2}$ instead of $\mathcal{M}_A = \frac{2}{3}$.

Case 2: link-b is no longer a system bottleneck; instead, link-b' ($\neq b$) is now one of the system bottlenecks.

We need to consider the following two sub-cases.

- If $D'^{b'} \leq D^{b}$, the progress in the first stage will increase; i.e., $\mathcal{M}'^{b'}_{k} \geq \mathcal{M}^{b}_{k}$. However, tenant-$k$'s allocation in link-$b$ will be no larger than if she had told the truth, making her progress no better. To see this, consider the allocations of all other tenants in link-b before and after she lies. Denote by c^{b}_{-k} and c'^{b}_{-k} the resource consumption of all other tenants in link-b when tenant-k was telling the truth and lying, respectively. We also have $c^{b}_{-k} = a^{b}_{-k}$ and $a^{b}_{-k} + a^{b}_{k} = 1$ because link-b was the bottleneck, and there was no spare resource to allocate for this link. When tenant-k lies, $a'^{b}_{-k} \geq a^{b}_{-k}$ because $\mathcal{M}'^{b'}_{k} \geq \mathcal{M}^{b}_{k}$. We also have $c'^{b}_{-k} \geq a'^{b}_{-k}$ and $c'^{b}_{-k} + c'^{b}_{k} \leq 1$. This implies $c'^{b}_{k} \leq 1 - c'^{b}_{-k} \leq 1 - a'^{b}_{-k} \leq 1 - a^{b}_{-k} = a^{b}_{k} = c^{b}_{k}$. Meaning, tenant-$k$'s progress is no larger than that when she was telling the truth.
- If $D'^{b'} > D^{b}$, everyone's progress including her own decreases in the first stage ($\mathcal{M}'^{b'}_{k} < \mathcal{M}^{b}_{k}$). Similar to the second scenario in Case 1, the second stage will restrict tenant-k to the lowered progress.

Regardless of tenant-k's approaches – keeping the same system bottleneck or not – her progress using Algorithm 1 will not increase. □

Corollary 4 (of Theorem 3) *Algorithm 1 maximizes isolation guarantee, i.e., the minimum progress across tenants.* □

Theorem 5 *Algorithm 1 achieves the highest resource utilization among all strategyproof algorithms that provide optimal isolation guarantee among tenants.*

Proof Sketch (of Theorem 5) Follows from Lemma 1 and Theorem 3. □

Lemma 6 *Under some cases, DRF may have utilization arbitrarily close to 0, and HUG helps improve the utilization to 1.*

Proof Sketch (of Lemma 6) Construct the cases with K links and N tenants, and each tenant has demand 1 on link-1 and ϵ on other links.

DRF will allocate to each tenant $\frac{1}{N}$ on link-1 and $\frac{\epsilon}{N}$ on all other links, resulting in a total utilization of $\frac{1+(K-1)\epsilon}{K} \to 0$ when $K \to \infty, \epsilon \to 0$ for any N.

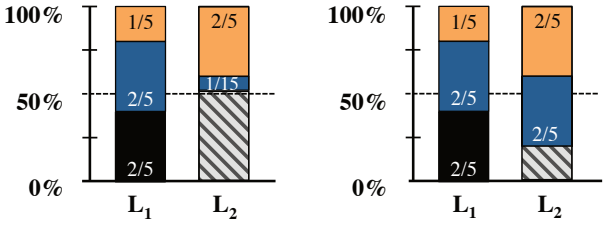


Figure 14: Hard tradeoff between work conservation and strategy-proofness. Adding one more tenant (tenant-C in black) to Figure 5 with correlation vector $\langle 1, 0 \rangle$ makes simultaneously achieving work conservation and optimal isolation guarantee impossible, even when all three have elastic demands.

HUG will allocate to each tenant $\frac{1}{N}$ on every link and achieve 100% utilization. □

B Tradeoff Between Work Conservation and Strategy-proofness

We demonstrate the tradeoff between work conservation and strategy-proofness (thus isolation guarantee) by extending our running example from Section 2.

Consider another tenant (tenant-C) with correlation vector $\overrightarrow{d_C} = \langle 1, 0 \rangle$ in addition to the two tenants present earlier. The key distinction between tenant-C and either of the earlier two is that she does not demand any bandwidth on link-2. Given the three correlation vectors, we can use DRF to calculate the optimal isolation guarantee (Figure 14a), where tenant-k has $\mathcal{M}_k = \frac{2}{5}$, link-1 is completely utilized, and $\frac{7}{15}$-th of link-2 is proportionally divided between tenant-A and tenant-B.

This leaves us with two questions:

1. *How do we completely allocate the remaining $\frac{8}{15}$-th bandwidth of link-2?*
2. *Is it even possible without sacrificing optimal isolation guarantee and strategy-proofness?*

We show in the following that it is indeed not possible to allocate more than $\frac{4}{5}$-th of link-2 (Figure 14b) without sacrificing the optimal isolation guarantee.

Let us consider three primary categories of work-conserving spare allocation policies: *demand-agnostic*, *unfair*, and *locally fair*. All three will result in lower isolation guarantee, lower utilization, or both.

B.1 Demand-Agnostic Policies

Demand-agnostic policies equally divide the resource between the number of tenants independently in each link, irrespective of tenant demands, and provide isolation. Although strategyproof, this allocation (Figure 15a) has lower isolation guarantee ($\mathcal{M}_A = \frac{1}{2}$ and $\mathcal{M}_B = \mathcal{M}_C = \frac{1}{3}$, therefore isolation guarantee is $\frac{1}{3}$) than the op-

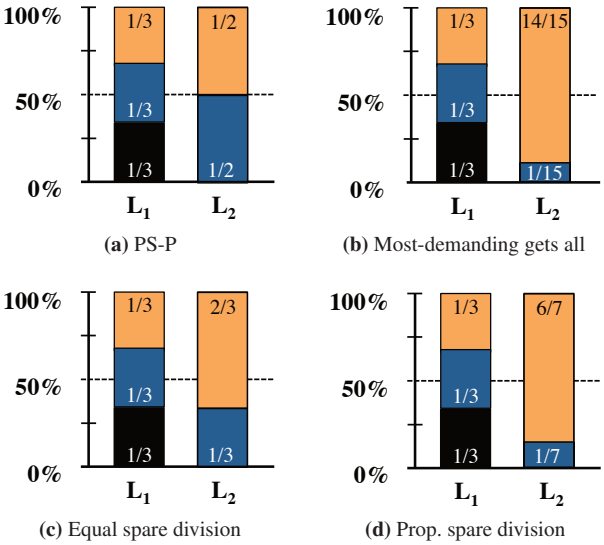


Figure 15: Allocations after applying different work-conserving policies to divide spare capacities in link-2 for the example in Figure 14.

timal isolation guarantee allocation shown in Figure 14a ($\mathcal{M}_A = \mathcal{M}_B = \mathcal{M}_C = \frac{2}{5}$, therefore isolation guarantee is $\frac{2}{5}$). PS-P [45, 58, 59] fall in this category.

Worse, when tenants do not have elastic-demand applications, demand-agnostic policies are not even work-conserving (similar to the example in §2.3.4).

Lemma 7 *When tenants do not have elastic demands, per-resource equal sharing is not work-conserving.*

Proof Sketch Only $\frac{11}{12}$-th of link-1 and $\frac{5}{9}$-th of link-2 will be consumed; i.e., none of the links will be saturated! □

To make it work-conserving, PS-P suggests dividing spare resources based on whoever wants it.

Lemma 8 *When tenants do not have elastic demands, PS-P is not work-conserving.*

Proof Sketch If tenant-B gives up her spare allocation in link-2, tenant-A can increase her progress to $\mathcal{M}_A = \frac{2}{3}$ and saturate link-1; however, tenant-B and tenant-C will remain at $\mathcal{M}_B = \mathcal{M}_C = \frac{1}{3}$. If tenant-$A$ gives up her spare allocation in link-1, tenant-B and tenant-C can increase their progress to $\mathcal{M}_B = \mathcal{M}_C = \frac{3}{8}$ and saturate link-1, but tenant-A will remain at $\mathcal{M}_A = \frac{1}{2}$. Because both tenant-$A$ and tenant-B have chances of increasing their progress, both will hold off to their allocations even with useless traffic – another instance of Prisoner's dilemma. □

B.2 Unfair Policies

Instead of demand-agnostic policies, one can also consider simpler, unfair policies; e.g., allocating all the resources to the tenant with the least or the most demand.

Lemma 9 *Allocating spare resource to the tenant with the least demand can result in zero spare allocation.*

Proof Sketch Although this strategy provides the optimal allocation for Figure 5, when at least one tenant in a link has zero demand, it can trivially result in no additional utilization; e.g., tenant-C in Figure 14. □

Lemma 10 *Allocating spare resource to the tenant with the least demand is not strategyproof.*

Proof Sketch Consider tenant-A lied and changed her correlation vector to $\overrightarrow{d'_A} = \langle 1, \frac{1}{10} \rangle$. The new optimal isolation guarantee allocation for unchanged tenant-B and tenant-C correlation vectors would be: $\overrightarrow{a_A} = \langle \frac{1}{3}, \frac{1}{30} \rangle$, $\overrightarrow{a_B} = \langle \frac{1}{3}, \frac{1}{18} \rangle$, and $\overrightarrow{a_C} = \langle \frac{1}{3}, 0 \rangle$. Now the spare resource in link-2 will be allocated to tenant-A because she asked for the least amount, and her final allocation would be $\overrightarrow{a'_A} = \langle \frac{1}{3}, \frac{17}{18} \rangle$. As a result, her progress improved from $\mathcal{M}_A = \frac{2}{5}$ to $\mathcal{M}'_A = \frac{2}{3}$, while the others' decreased to $\mathcal{M}_B = \mathcal{M}_C = \frac{1}{3}$. □

Corollary 11 (of Lemma 10) *In presence of work conservation, tenants can lie both by increasing and decreasing their demands, or a combination of both.* □

Lemma 12 *Allocating spare resource to the tenant with the highest demand is not strategyproof.*

Proof Sketch If tenant-A changes her correlation vector to $\overrightarrow{d'_A} = \langle 1, 1 \rangle$, the eventual allocation (Figure 15b) will again result in lower progress ($\mathcal{M}_B = \mathcal{M}_C = \frac{1}{3}$). Because tenant-$B$ is still receiving more than $\frac{1}{6}$-th of her allocation in link-1 in link-2, she does not need to lie. □

Corollary 13 (of Lemmas 10, 12) *Allocating spare resource randomly to tenants is not strategyproof.* □

B.3 Locally Fair Policies

Finally, one can also consider equally or proportionally dividing the spare resource on link-2 between tenant-A and tenant-B. Unfortunately, these strategies are not strategyproof either.

Lemma 14 *Allocating spare resource equally to tenants is not strategyproof.*

Proof Sketch If the remaining $\frac{8}{15}$-th of link-2 is equally divided, the share of tenant-A will increase to $\frac{2}{3}$-rd and incentivize her to lie. Again, the isolation guarantee will be smaller (Figure 15c). □

Lemma 15 *Allocating spare resource proportionally to tenants' demands is not strategyproof.*

Proof Sketch If one divides the spare in proportion to tenant demands, the allocation is different (Figure 15d) than equal division. However, tenant-A can again increase her progress at the expense of others. □

C Dual Objectives of Network Sharing

The two conflicting requirements of the network sharing problem can be defined as follows.

1. Utilization: $\sum_{i\in[1,2P]}\sum_{k\in[1,M]} c_k^i$
2. Isolation guarantee: $\min_{k\in[1,M]} \mathcal{M}_k$

Given the tradeoff between the two, one can consider one of the two possible optimizations:[8]

O1 Ensure highest utilization, *then* maximize the isolation guarantee with best effort;

O2 Ensure optimal isolation guarantee, *then* maximize utilization with best effort.

O1: Utilization-First In this case, the optimization attempts to maximize the isolation guarantee across all tenants while keeping the highest network utilization.

$$\begin{aligned} \text{Maximize} \quad & \min_{k\in[1,M]} \mathcal{M}_k \\ \text{s.t.} \quad & \sum_{i\in[1,2P]}\sum_{k\in[1,M]} c_k^i = U^*, \end{aligned} \tag{3}$$

where $U^* = \max \sum_{i\in[1,2P]}\sum_{k\in[1,M]} c_k^i$ is the highest utilization possible. Although this ensures maximum network utilization, isolation guarantee to individual tenants can be arbitrarily low. This formulation can still be useful in private datacenters [36].

To ensure some isolation guarantee, existing cloud network sharing approaches [14, 45, 51, 58, 59, 61, 62, 67] use a similar formulation:

$$\begin{aligned} \text{Maximize} \quad & \sum_{1\le i\le 2P}\sum_{k\in[1,M]} c_k^i \\ \text{subject to} \quad & \mathcal{M}_k \ge \frac{1}{M},\ k\in[1,M] \end{aligned} \tag{4}$$

The objective here is to maximize utilization while ensuring at least $\frac{1}{M}$-th of each link to tenant-k. However, this approach has two primary drawbacks (§ 2.3):

1. suboptimal isolation guarantee, and
2. lower utilization.

O2: Isolation-Guarantee-First Instead, in this paper, we have formulated the network sharing problem as follows:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i\in[1,2P]}\sum_{k\in[1,M]} c_k^i \\ \text{subject to} \quad & \min_{k\in[1,M]} \mathcal{M}_k = \mathcal{M}_k^* \\ & c_k^i \ge a_k^i,\ i\in[1,2P], k\in[1,M] \end{aligned} \tag{5}$$

Here, we maximize resource consumption while keeping the optimal isolation guarantee across all tenants, denoted by $\mathcal{M}_k^*$. Meanwhile, the constraint on consumption being at least guaranteed minimum allocation ensures strategy-proofness; thus, guaranteeing that guaranteed allocated resources will be utilized.

Because c_k^i values have no upper bounds except for physical capacity constraints, optimization **O2** may result in suboptimal isolation guarantee in non-cooperative environments (§2.3.3). HUG introduces the following additional constraint to avoid this issue only in non-cooperative environments:

$$c_k^i \le \mathcal{M}^*,\ i\in[1,2P], k\in[1,M]$$

This constraint is not necessary when strategy-proofness is a non-requirement – e.g., in private datacenters.

[8] Maximizing a combination of these two is also an interesting future direction.