Dominant Resource Fairness in Cloud Computing Systems with Heterogeneous Servers

Wei Wang, Baochun Li, Ben Liang
Department of Electrical and Computer Engineering
University of Toronto

***Abstract*—We study the multi-resource allocation problem in cloud computing systems where the resource pool is constructed from a large number of heterogeneous servers, representing different points in the configuration space of resources such as processing, memory, and storage. We design a multi-resource allocation mechanism, called DRFH, that generalizes the notion of Dominant Resource Fairness (DRF) from a single server to multiple heterogeneous servers. DRFH provides a number of highly desirable properties. With DRFH, no user prefers the allocation of another user; no one can improve its allocation without decreasing that of the others; and more importantly, no user has an incentive to lie about its resource demand. As a direct application, we design a simple heuristic that implements DRFH in real-world systems. Large-scale simulations driven by Google cluster traces show that DRFH significantly outperforms the traditional slot-based scheduler, leading to much higher resource utilization with substantially shorter job completion times.**

I. Introduction

Resource allocation under the notion of fairness and efficiency is a fundamental problem in the design of cloud computing systems. Unlike traditional application-specific clusters and grids, a cloud computing system distinguishes itself with unprecedented server and workload heterogeneity. Modern datacenters are likely to be constructed from a variety of server classes, with different configurations in terms of processing capabilities, memory sizes, and storage spaces [1]. Asynchronous hardware upgrades, such as adding new servers and phasing out existing ones, further aggravate such diversity, leading to a wide range of server specifications in a cloud computing system [2]. Table I illustrates the heterogeneity of servers in one of Google's clusters [2], [3].

In addition to server heterogeneity, cloud computing systems also represent much higher diversity in resource demand profiles. Depending on the underlying applications, the workload spanning multiple cloud users may require vastly different amounts of resources (*e.g.*, CPU, memory, and storage). For example, numerical computing tasks are usually CPU intensive, while database operations typically require high-memory support. The heterogeneity of both servers and workload demands poses significant technical challenges on the resource allocation mechanism, giving rise to many delicate issues — notably fairness and efficiency — that must be carefully addressed.

Despite the unprecedented heterogeneity in cloud computing systems, state-of-the-art computing frameworks employ rather simple abstractions that fall short. For example, Hadoop [4] and Dryad [5], the two most widely deployed cloud computing frameworks, partition a server's resources into bundles — known as *slots* — that contain fixed amounts of different resources. The system then allocates resources to users at the granularity of these slots. Such a *single resource* abstraction ignores the heterogeneity of both server specifications and demand profiles, inevitably leading to a fairly inefficient allocation [6].

TABLE I
Configurations of servers in one of Google's clusters [2], [3]. CPU and memory units are normalized to the maximum server (highlighted below).

Number of servers	CPUs	Memory
6732	0.50	0.50
3863	0.50	0.25
1001	0.50	0.75
795	**1.00**	**1.00**
126	0.25	0.25
52	0.50	0.12
5	0.50	0.03
5	0.50	0.97
3	1.00	0.50
1	0.50	0.06

Towards addressing the inefficiency of the current allocation system, many recent works focus on *multi-resource allocation* mechanisms. Notably, Ghodsi *et al.* [6] suggest a compelling alternative known as the Dominant Resource Fairness (DRF) allocation, in which each user's *dominant share* — the maximum ratio of any resource that the user has been allocated in a server — is equalized. The DRF allocation possesses a set of highly desirable fairness properties, and has quickly received significant attention in the literature [7], [8], [9], [10]. While DRF and its subsequent works address the demand heterogeneity of multiple resources, they all limit the discussions to a simplified model where all resources are concentrated into one super computer[1]. Such an *all-in-one* resource model drastically contrasts the state-of-the-practice infrastructure of cloud computing systems. Moreover, it ignores the heterogeneity of servers: The resulting resource allocations only depend on the total amount of resources pooled in the system, irrespective of the underlying resource distribution of servers. In fact, with heterogeneous servers, even the definition of *dominant resource* is unclear: Depending on the underlying server configurations, a computing task may bottleneck on different resources in different servers. We shall note that naive extensions, such as applying the DRF allocation to each server separately, leads to a highly inefficient allocation (details in Sec. III-D).

[1]While [6] briefly touches on the case where resources are distributed to small servers (known as the discrete scenario), its coverage is rather informal.

978-1-4799-3360-0/14/$31.00

This paper represents the first rigorous study to propose a solution with provable operational benefits that bridge the gap between the existing multi-resource allocation models and the prevalent datacenter infrastructure. We propose DRFH, a *DRF* generalization in *Heterogeneous* environments where resources are pooled by a large amount of heterogeneous servers, representing different points in the configuration space of resources such as processing, memory, and storage. DRFH generalizes the intuition of DRF by seeking an allocation that equalizes every user's *global dominant share*, which is the maximum ratio of any resources the user has been allocated in the *entire* cloud resource pool. We systematically analyze DRFH and show that it retains most of the desirable properties that the all-in-one DRF model provides [6]. Specifically, DRFH is *Pareto optimal*, where no user is able to increase its allocation without decreasing other users' allocations. Meanwhile, DRFH is *envy-free* in that no user prefers the allocation of another user. More importantly, DRFH is *truthful* in that a user cannot schedule more computing tasks by claiming more resources that are not needed, and hence has no incentive to misreport its actual resource demand. DRFH also satisfies a set of other important properties, namely *single-server DRF*, *single-resource fairness*, *bottleneck fairness*, and *population monotonicity* (details in Sec. III-C).

As a direct application, we design a heuristic scheduling algorithm that implements DRFH in real-world systems. We conduct large-scale simulations driven by Google cluster traces [3]. Our simulation results show that compared to the traditional slot schedulers adopted in prevalent cloud computing frameworks, the DRFH algorithm suitably matches demand heterogeneity to server heterogeneity, significantly improving the system's resource utilization, yet with a substantial reduction of job completion times.

II. Related Work

Despite the extensive computing system literature on fair resource allocation, many existing works limit their discussions to the allocation of a single resource type, *e.g.*, CPU time [11], [12] and link bandwidth [13], [14], [15], [16], [17]. Various fairness notions have also been proposed throughout the years, ranging from application-specific allocations [18], [19] to general fairness measures [13], [20], [21].

As for multi-resource allocation, state-of-the-art cloud computing systems employ naive single resource abstractions. For example, the two fair sharing schedulers currently supported in Hadoop [22], [23] partition a node into slots with fixed fractions of resources, and allocate resources jointly at the slot granularity. Quincy [24], a fair scheduler developed for Dryad [5], models the fair scheduling problem as a min-cost flow problem to schedule jobs into slots. The recent work [25] takes the job placement constraints into consideration, yet it still uses a slot-based single resource abstraction.

Ghodsi *et al.* [6] are the first in the literature to present a systematic investigation on the multi-resource allocation problem in cloud computing systems. They propose DRF to equalize the dominant share of all users, and show that a number of desirable fairness properties are guaranteed in the resulting allocation. DRF has quickly attracted a substantial amount of attention and has been generalized to many dimensions. Notably, Joe-Wong *et al.* [7] generalize the DRF measure and incorporate it into a unifying framework that captures the trade-offs between allocation fairness and efficiency. Dolev *et al.* [8] suggest another notion of fairness for multi-resource allocation, known as Bottleneck-Based Fairness (BBF), under which two fairness properties that DRF possesses are also guaranteed. Gutman and Nisan [9] consider another settings of DRF with a more general domain of user utilities, and show their connections to the BBF mechanism. Parkes *et al.* [10], on the other hand, extend DRF in several ways, including the presence of zero demands for certain resources, weighted user endowments, and in particular the case of indivisible tasks. They also study the loss of social welfare under the DRF rules. More recently, Kash *et al.* [26] extend the DRF model to allow users to join the system over time but will never leave. However, all these works assume, explicitly or implicitly, that all resources are concentrated into a super computer, which is not the case in the prevalent datacenter system.

Other related works include fair-division problems in the economics literature, in particular the *egalitarian division* under Leontief preferences [27] and the *cake-cutting problem* [28]. These works also assume the *all-in-one* resource model, and hence cannot be directly applied to cloud computing systems with heterogeneous servers.

III. System Model and Allocation Properties

In this section, we model multi-resource allocation in a cloud computing system with heterogeneous servers. We formalize a number of desirable properties that are deemed the most important for allocation mechanisms in cloud computing environments.

A. Basic Setting

In a cloud computing system, the resource pool is composed of a cluster of heterogeneous servers $S = \{1, \ldots, k\}$, each contributing m hardware resources (*e.g.*, CPU, memory, storage) denoted by $R = \{1, \ldots, m\}$. For each server l, let $\mathbf{c}_l = (c_{l1}, \ldots, c_{lm})^T$ be its *resource capacity vector*, where each element c_{lr} denotes the total amount of resource r available in server l. Without loss of generality, for every resource r, we normalize the total capacity of all servers to 1, *i.e.*,

$$\sum_{l \in S} c_{lr} = 1, \quad r = 1, 2, \ldots, m.$$

Let $U = \{1, \ldots, n\}$ be the set of cloud users sharing the cloud system. For every user i, let $\mathbf{D}_i = (D_{i1}, \ldots, D_{im})^T$ be its *resource demand vector*, where D_{ir} is the fraction (share) of resource r required by each task of user i over the *entire* system. For simplicity, we assume positive demands for all users, *i.e.*, $D_{ir} > 0, \forall i \in U, r \in R$. We say resource r_i^* is the *global dominant resource* of user i if

$$r_i^* \in \arg\max_{r \in R} D_{ir} \ .$$

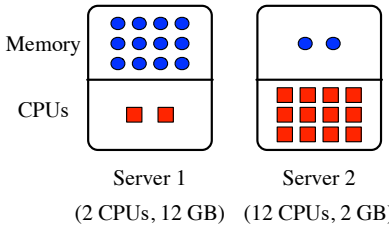


Fig. 1. An example of a system containing two heterogeneous servers shared by two users. Each computing task of user 1 requires 0.2 CPU time and 1 GB memory, while the computing task of user 2 requires 1 CPU time and 0.2 GB memory.

In other words, r_i^* is the most heavily demanded resource required by user i's task in the entire resource pool. For all user i and resource r, we define

$$d_{ir} = D_{ir}/D_{ir_i^*}$$

as the *normalized demand* and denote by $\mathbf{d}_i = (d_{i1}, \ldots, d_{im})^T$ the normalized demand vector of user i.

As a concrete example, consider Fig. 1 where the system contains two heterogeneous servers. Server 1 is high-memory with 2 CPUs and 12 GB memory, while server 2 is high-CPU with 12 CPUs and 2 GB memory. Since the system contains 14 CPUs and 14 GB memory in total, the normalized capacity vectors of server 1 and 2 are $\mathbf{c}_1 = (\text{CPU share}, \text{memory share})^T = (1/7, 6/7)^T$ and $\mathbf{c}_2 = (6/7, 1/7)^T$, respectively. Now suppose there are two users. User 1 has memory-intensive tasks each requiring 0.2 CPU time and 1 GB memory, while user 2 has CPU-heavy tasks each requiring 1 CPU time and 0.2 GB memory. The demand vector of user 1 is $\mathbf{D}_1 = (1/70, 1/14)^T$ and the normalized vector is $\mathbf{d}_1 = (1/5, 1)^T$, where memory is the global dominant resource. Similarly, user 2 has $\mathbf{D}_2 = (1/14, 1/70)^T$ and $\mathbf{d}_2 = (1, 1/5)^T$, and CPU is its global dominant resource.

For now, we assume users have an infinite number of tasks to be scheduled, and all tasks are divisible [6], [8], [9], [10], [26]. We shall discuss how these assumptions can be relaxed in Sec. V.

B. Resource Allocation

For every user i and server l, let $\mathbf{A}_{il} = (A_{il1}, \ldots, A_{ilm})^T$ be the *resource allocation vector*, where A_{ilr} is the share of resource r allocated to user i in server l. Let $\mathbf{A}_i = (\mathbf{A}_{i1}, \ldots, \mathbf{A}_{ik})$ be the *allocation matrix* of user i, and $\mathbf{A} = (\mathbf{A}_1, \ldots, \mathbf{A}_n)$ the overall allocation for all users. We say an allocation $\mathbf{A}$ is *feasible* if no server is required to use more than any of its total resources, *i.e.*,

$$\sum_{i \in U} A_{ilr} \le c_{lr}, \quad \forall l \in S, r \in R \ .$$

For all user i, given allocation $\mathbf{A}_{il}$ in server l, the maximum number of tasks (possibly fractional) that it can schedule is calculated as

$$N_{il}(\mathbf{A}_{il}) = \min_{r \in R}\{A_{ilr}/D_{ir}\} \ .$$

The total number of tasks user i can schedule under allocation $\mathbf{A}_i$ is hence

$$N_i(\mathbf{A}_i) = \sum_{l \in S} N_{il}(\mathbf{A}_{il}) \ . \tag{1}$$

Intuitively, a user prefers an allocation that allows it to schedule more tasks.

A well-justified allocation should never give a user more resources than it can actually use in a server. Following the terminology used in the economics literature [27], we call such an allocation *non-wasteful*:

Definition 1: For user i and server l, an allocation $\mathbf{A}_{il}$ is *non-wasteful* if taking out any resources reduces the number of tasks scheduled, *i.e.,* for all $\mathbf{A}'_{il} \prec \mathbf{A}_{il}$[2], we have that

$$N_{il}(\mathbf{A}'_{il}) < N_{il}(\mathbf{A}_{il}) \ .$$

User i's allocation $\mathbf{A}_i = (\mathbf{A}_{il})$ is non-wasteful if $\mathbf{A}_{il}$ is non-wasteful for all server l, and allocation $\mathbf{A} = (\mathbf{A}_i)$ is non-wasteful if $\mathbf{A}_i$ is non-wasteful for all user i.

Note that one can always convert an allocation to non-wasteful by revoking those resources that are allocated but have never been actually used, without changing the number of tasks scheduled for any user. Therefore, unless otherwise specified, we limit the discussions to non-wasteful allocations.

C. Allocation Mechanism and Desirable Properties

A resource allocation mechanism takes user demands as input and outputs the allocation result. In general, an allocation mechanism should provide the following essential properties that are widely recognized as the most important fairness and efficiency measures in both cloud computing systems [6], [7], [25] and the economics literature [27], [28].

Envy-freeness: An allocation mechanism is *envy-free* if no user prefers the other's allocation to its own, *i.e.*, $N_i(\mathbf{A}_i) \ge N_i(\mathbf{A}_j)$ for any two users $i, j \in U$. This property essentially embodies the notion of fairness.

Pareto optimality: An allocation mechanism is *Pareto optimal* if it returns an allocation $\mathbf{A}$ such that for all feasible allocations $\mathbf{A}'$, if $N_i(\mathbf{A}'_i) > N_i(\mathbf{A}_i)$ for some user i, then there exists a user j such that $N_j(\mathbf{A}'_j) < N_j(\mathbf{A}_j)$. In other words, there is no other allocation where all users are at least as well off and at least one user is strictly better off. This property ensures the allocation efficiency and is critical for high resource utilization.

Truthfulness: An allocation mechanism is *truthful* if no user can schedule more tasks by misreporting its resource demand (assuming a user's demand is its private information), irrespective of other users' behaviour. Specifically, given the demands claimed by other users, let $\mathbf{A}$ be the resulting allocation when user i truthfully reports its resource demand $\mathbf{D}_i$, and let $\mathbf{A}'$ be the allocation returned when user i misreports by $\mathbf{D}'_i \neq \mathbf{D}_i$. Then under a truthful mechanism we have $N_i(\mathbf{A}_i) \ge N_i(\mathbf{A}'_i)$. Truthfulness is of a special importance for a cloud computing system, as it is common to observe in real-world systems that users try to lie about their resource demands to manipulate the schedulers for more allocation [6], [25].

[2]For any two vectors $\mathbf{x}$ and $\mathbf{y}$, we say $\mathbf{x} \prec \mathbf{y}$ if $x_i \le y_i, \forall i$ and for some j we have strict inequality: $x_j < y_j$.

In addition to these essential properties, we also consider four other important properties below:

Single-server DRF: If the system contains only one server, then the resulting allocation should be reduced to the DRF allocation.

Single-resource fairness: If there is a single resource in the system, then the resulting allocation should be reduced to a max-min fair allocation.

Bottleneck fairness: If all users bottleneck on the same resource (*i.e.,* having the same global dominant resource), then the resulting allocation should be reduced to a max-min fair allocation for that resource.

Population monotonicity: If a user leaves the system and relinquishes all its allocations, then the remaining users will not see any reduction in the number of tasks scheduled.

In addition to the aforementioned properties, *sharing incentive* is another important property that has been frequently mentioned in the literature [6], [7], [8], [10]. It ensures that every user's allocation is not worse off than that obtained by evenly dividing the entire resource pool. While this property is well defined for a single server, it is not for a system containing multiple heterogeneous servers, as there is an infinite number of ways to evenly divide the resource pool among users, and it is unclear which one should be chosen as a benchmark. We defer the discussions to Sec. IV-D, where we justify between two possible alternatives. For now, our objective is to design an allocation mechanism that guarantees all the properties defined above.

D. Naive DRF Extension and Its Inefficiency

It has been shown in [6], [10] that the DRF allocation satisfies all the desirable properties mentioned above when there is only one server in the system. The key intuition is to equalize the fraction of dominant resources allocated to each user in the server. When resources are distributed to many heterogeneous servers, a naive generalization is to separately apply the DRF allocation per server. Since servers are heterogeneous, a user might have different dominant resources in different servers. For instance, in the example of Fig. 1, user 1's dominant resource in server 1 is CPU, while its dominant resource in server 2 is memory. Now apply DRF in server 1. Because CPU is also user 2's dominant resource, the DRF allocation lets both users have an equal share of the server's CPUs, each allocated 1. As a result, user 1 schedules 5 tasks onto server 1, while user 2 schedules 1 onto the same server. Similarly, in server 2, memory is the dominant resource of both users and is evenly allocated, leading to 1 task scheduled for user 1 and 5 for user 2. The resulting allocations in the two servers are illustrated in Fig. 2, where both users schedule 6 tasks.

Unfortunately, this allocation violates Pareto optimality and is highly inefficient. If we instead allocate server 1 exclusively to user 1, and server 2 exclusively to user 2, then both users schedule 10 tasks, more than those scheduled under the DRF allocation. In fact, we see that naively applying DRF per server may lead to an allocation with arbitrarily low resource utilization.

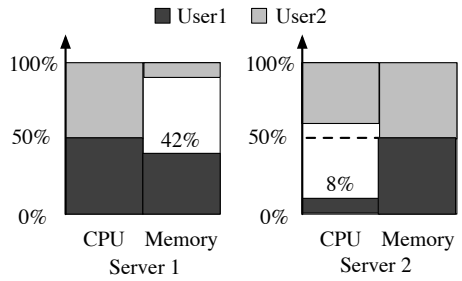


Fig. 2. DRF allocation for the example shown in Fig. 1, where user 1 is allocated 5 tasks in server 1 and 1 in server 2, while user 2 is allocated 1 task in server 1 and 5 in server 2.

The failure of the naive DRF extension to the heterogeneous environment necessitates an alternative allocation mechanism, which is the main theme of the next section.

IV. DRFH Allocation and Its Properties

In this section, we describe DRFH, a generalization of DRF in a heterogeneous cloud computing system where resources are distributed in a number of heterogeneous servers. We analyze DRFH and show that it provides all the desirable properties defined in Sec. III.

A. DRFH Allocation

Instead of allocating separately in each server, DRFH jointly considers resource allocation across all heterogeneous servers. The key intuition is to achieve the *max-min fair allocation for the global dominant resources*. Specifically, given allocation $\mathbf{A}_{il}$, let $G_{il}(\mathbf{A}_{il})$ be the fraction of global dominant resources user i receives in server l, *i.e.,*

$$G_{il}(\mathbf{A}_{il}) = N_{il}(\mathbf{A}_{il})D_{ir_i^*} = \min_{r\in R}\{A_{ilr}/d_{ir}\} \ . \tag{2}$$

We call $G_{il}(\mathbf{A}_{il})$ the *global dominant share* user i receives in server l under allocation $\mathbf{A}_{il}$. Therefore, given the overall allocation $\mathbf{A}_i$, the global dominant share user i receives is

$$G_i(\mathbf{A}_i) = \sum_{l\in S} G_{il}(\mathbf{A}_{il}) = \sum_{l\in S} \min_{r\in R}\{A_{ilr}/d_{ir}\} \ . \tag{3}$$

DRFH allocation aims to maximize the minimum global dominant share among all users, subject to the resource constraints per server, *i.e.,*

$$\begin{aligned} \max_{\mathbf{A}} \quad & \min_{i\in U} G_i(\mathbf{A}_i) \\ \text{s.t.} \quad & \sum_{i\in U} A_{ilr} \le c_{lr}, \forall l \in S, r \in R \ . \end{aligned} \tag{4}$$

Recall that without loss of generality, we assume non-wasteful allocation $\mathbf{A}$ (see Sec. III-B). We have the following structural result. The proof is deferred to our technical report [29].

Lemma 1: For user i and server l, an allocation $\mathbf{A}_{il}$ is non-wasteful *if and only if* there exists some g_{il} such that $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$. In particular, g_{il} is the global dominant share user i receives in server l under allocation $\mathbf{A}_{il}$, *i.e.,*

$$g_{il} = G_{il}(\mathbf{A}_{il}) \ .$$

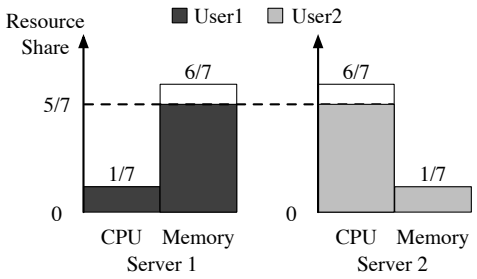


Fig. 3. An alternative allocation with higher system utilization for the example of Fig. 1. Server 1 and 2 are exclusively assigned to user 1 and 2, respectively. Both users schedule 10 tasks.

Intuitively, Lemma 1 indicates that under a non-wasteful allocation, resources are allocated *in proportion* to the user's demand. Lemma 1 immediately suggests the following relationship for all user i and its non-wasteful allocation $\mathbf{A}_i$:

$$G_i(\mathbf{A}_i) = \sum_{l\in S} G_{il}(\mathbf{A}_{il}) = \sum_{l\in S} g_{il} \ .$$

Problem (4) can hence be equivalently written as

$$\begin{aligned} \max_{\{g_{il}\}} \quad & \min_{i\in U} \sum_{l\in S} g_{il} \\ \text{s.t.} \quad & \sum_{i\in U} g_{il} d_{ir} \le c_{lr}, \forall l \in S, r \in R \ , \end{aligned} \tag{5}$$

where the constraints are derived from Lemma 1. Now let $g = \min_i \sum_{l\in S} g_{il}$. Via straightforward algebraic operation, we see that (5) is equivalent to the following problem:

$$\begin{aligned} \max_{\{g_{il}\}} \quad & g \\ \text{s.t.} \quad & \sum_{i\in U} g_{il} d_{ir} \le c_{lr}, \forall l \in S, r \in R \ , \\ & \sum_{l\in U} g_{il} = g, \forall i \in U \ . \end{aligned} \tag{6}$$

Note that the second constraint ensures the fairness with respect to the equalized global dominant share g. By solving (6), DRFH allocates each user the maximum global dominant share g, under the constraints of both server capacity and fairness. By Lemma 1, the allocation received by each user i in server l is simply $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$.

For example, Fig. 3 illustrates the resulting DRFH allocation in the example of Fig. 1. By solving (6), DRFH allocates server 1 exclusively to user 1 and server 2 exclusively to user 2, allowing each user to schedule 10 tasks with the maximum global dominant share $g = 5/7$.

We next analyze the properties of DRFH allocation obtained by solving (6) in the following two subsections.

B. Analysis of Essential Properties

Our analysis of DRFH starts with the three essential resource allocation properties, namely, envy-freeness, Pareto optimality, and truthfulness. We first show that under the DRFH allocation, no user prefers other's allocation to its own.

Proposition 1 (Envy-freeness): The DRFH allocation obtained by solving (6) is envy-free.

Proof: Let $\{g_{il}\}$ be the solution to problem (6). For all user i, its DRFH allocation in server l is $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$. To show $N_i(\mathbf{A}_j) \le N_i(\mathbf{A}_i)$ for any two users i and j, it is equivalent to prove $N_i(\mathbf{A}_j) \le N_i(\mathbf{A}_i)$. We have

$$\begin{aligned} G_i(\mathbf{A}_j) &= \textstyle\sum_l G_{il}(\mathbf{A}_{jl}) \\ &= \textstyle\sum_l \min_r\{g_{jl}d_{jr}/d_{ir}\} \\ &\le \textstyle\sum_l g_{jl} = G_i(\mathbf{A}_i) \ , \end{aligned}$$

where the inequality holds because

$$\min_r\{d_{jr}/d_{ir}\} \le d_{jr_i^*}/d_{ir_i^*} \le 1 \ ,$$

where r_i^* is user i's global dominant resource. ■

We next show that DRFH leads to an efficient allocation under which no user can improve its allocation without decreasing that of the others.

Proposition 2 (Pareto optimality): The DRFH allocation obtained by solving (6) is Pareto optimal.

Proof: Let $\{g_{il}\}$, and the corresponding g, be the solution to problem (6). For all user i, its DRFH allocation in server l is $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$. Since (5) and (6) are equivalent, $\{g_{il}\}$ also solves (5), with g being the maximum value of the objective of (5).

Assume, by way of contradiction, that allocation $\mathbf{A}$ is not Pareto optimal, *i.e.,* there exists some allocation $\mathbf{A}'$, such that $N_i(\mathbf{A}'_i) \ge N_i(\mathbf{A}_i)$ for all user i, and for some user j we have strict inequality: $N_j(\mathbf{A}'_j) > N_j(\mathbf{A}_j)$. Equivalently, this implies that $G_i(\mathbf{A}'_i) \ge G_i(\mathbf{A}_i)$ for all user i, and $G_j(\mathbf{A}'_j) > G_j(\mathbf{A}_j)$ for user j. Without loss of generality, let $\mathbf{A}'$ be non-wasteful. By Lemma 1, for all user i and server l, there exists some g'_{il} such that $\mathbf{A}'_{il} = g'_{il}\mathbf{d}_i$. We show that based on $\{g'_{il}\}$, one can construct some $\{\hat{g}_{il}\}$ such that $\{\hat{g}_{il}\}$ is a feasible solution to (5), yet leads to a higher objective than g, contradicting the fact that $\{g_{il}\}$ optimally solve (5).

To see this, consider user j. We have

$$G_j(\mathbf{A}_j) = \textstyle\sum_l g_{jl} = g < G_j(\mathbf{A}'_j) = \sum_l g'_{jl}.$$

For user j, there exists a server l_0 and some $\epsilon > 0$, such that after reducing g'_{jl_0} to $g'_{jl_0} - \epsilon$, the resulting global dominant share remains higher than g, *i.e.,* $\sum_l g'_{jl} - \epsilon \ge g$. This leads to at least $\epsilon\mathbf{d}_j$ idle resources in server l_0. We construct $\{\hat{g}_{il}\}$ by redistributing these idle resources to all users.

Denote by $\{g''_{il}\}$ the dominant share after reducing g'_{jl_0} to $g'_{jl_0} - \epsilon$, *i.e.,*

$$g''_{il} = \begin{cases} g'_{jl_0} - \epsilon, & i = j, l = l_0; \\ g'_{il}, & o.w. \end{cases}$$

The corresponding non-wasteful allocation is $\mathbf{A}''_{il} = g''_{il}\mathbf{d}_i$ for all user i and server l. Note that allocation $\mathbf{A}''$ is preferred over the original allocation $\mathbf{A}$ by all users, *i.e.,* for all user i, we have

$$G_i(\mathbf{A}''_i) = \sum_l g''_{il} = \begin{cases} \sum_l g'_{jl} - \epsilon \ge g = G_j(\mathbf{A}_j), & i = j; \\ \sum_l g'_{il} = G_i(\mathbf{A}'_i) \ge G_i(\mathbf{A}_i), & o.w. \end{cases}$$

We now construct $\{\hat{g}_{il}\}$ by redistributing the $\epsilon\mathbf{d}_j$ idle resources in server l_0 to all users, each increasing its global dominant share g''_{il_0} by $\delta = \min_r\{\epsilon d_{jr}/\sum_i d_{ir}\}$, *i.e.,*

$$\hat{g}_{il} = \begin{cases} g''_{il_0} + \delta, & l = l_0; \\ g''_{il}, & o.w. \end{cases}$$

It is easy to check that $\{\hat{g}_{il}\}$ remains a feasible allocation. To see this, it suffices to check server l_0. For all its resource r, we have

$$\begin{aligned}\textstyle\sum_i \hat{g}_{il_0} d_{ir} &= \textstyle\sum_i (g''_{il_0} + \delta) d_{ir} \\ &= \textstyle\sum_i g'_{il_0} d_{ir} - \epsilon d_{jr} + \delta \sum_i d_{ir} \\ &\leq c_{l_0 r} - (\epsilon d_{jr} - \delta \textstyle\sum_i d_{ir}) \leq c_{l_0 r} \ .\end{aligned}$$

where the first inequality holds because $\mathbf{A}'$ is a feasible allocation.

On the other hand, for all user $i \in U$, we have

$$\textstyle\sum_l \hat{g}_{il} = \sum_l g''_{il} + \delta = G_i(\mathbf{A}''_i) + \delta \geq G_i(\mathbf{A}_i) + \delta > g \ .$$

This contradicts the premise that g is optimal for (5). ■

For now, all our discussions are based on a critical assumption that all users truthfully report their resource demands. However, in a real-world system, it is common to observe users to attempt to manipulate the scheduler by misreporting their resource demands, so as to receive more allocation [6], [25]. More often than not, these strategic behaviours would significantly hurt those honest users and reduce the number of their tasks scheduled, inevitably leading to a fairly inefficient allocation outcome. Fortunately, we show by the following proposition that DRFH is immune to these strategic behaviours, as reporting the true demand is always the best strategy for every user, irrespective of the others' behaviour.

Proposition 3 (Truthfulness): The DRFH allocation obtained by solving (6) is truthful.

Proof: For any user i, fixing all other users' clamed demands $\mathbf{d}'_{-i} = (\mathbf{d}'_1, \ldots, \mathbf{d}'_{i-1}, \mathbf{d}'_{i+1}, \ldots, \mathbf{d}'_n)$ (which may not be their true demands), let $\mathbf{A}$ be the resulting allocation when i truthfully reports its demand $\mathbf{d}_i$, that is, $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$ and $\mathbf{A}_{jl} = g_{jl}\mathbf{d}'_j$ for all user $j \neq i$ and server l, where g_{il} and g_{jl} are the global dominant shares users i and j receive on server l under $\mathbf{A}_{il}$ and $\mathbf{A}_{jl}$, respectively. Similarly, let $\mathbf{A}'$ be the resulting allocation when user i misreports its demand as $\mathbf{d}'_i$. Let g and g' be the global dominant share user i receives under $\mathbf{A}_i$ and $\mathbf{A}'_i$, respectively. We check the following two cases and show that $G_i(\mathbf{A}'_i) \leq G_i(\mathbf{A}_i)$, which is equivalent to $N_i(\mathbf{A}'_i) \leq N_i(\mathbf{A}_i)$.

Case 1: $g' \leq g$. In this case, let $\rho_i = \min_r\{d'_{ir}/d_{ir}\}$ be defined for user i. Clearly,

$$\rho_i = \min_r\{d'_{ir}/d_{ir}\} \leq d'_{ir^*_i}/d_{ir^*_i} \leq 1 \ ,$$

where r^*_i is the dominant resource of user i. We then have

$$G_i(\mathbf{A}'_i) = \textstyle\sum_l G_{il}(\mathbf{A}'_{il}) = \rho_i g' \leq g = G_i(\mathbf{A}_i) \ .$$

Case 2: $g' > g$. For all user $j \neq i$, when user i truthfully reports its demand, let $G_j(\mathbf{A}_j, \mathbf{d}'_j)$ be the global dominant share of user j w.r.t. its *claimed demand* $\mathbf{d}'_j$, *i.e.,*

$$G_j(\mathbf{A}_j, \mathbf{d}'_j) = \textstyle\sum_l \min_r\{g_{jl}d'_{jr}/d'_{jr}\} = \sum_l g_{jl} = g \ .$$

Similarly, when user i misreports, let $G_j(\mathbf{A}'_j, \mathbf{d}'_j)$ be the global dominant share of user j w.r.t. its claimed demand $\mathbf{d}'_j$, *i.e.,*

$$G_j(\mathbf{A}'_j, \mathbf{d}'_j) = \textstyle\sum_l \min_r\{g'_{jl}d'_{jr}/d'_{jr}\} = \sum_l g'_{jl} = g' \ ,$$

As a result, $G_j(\mathbf{A}'_j, \mathbf{d}'_j) > G_j(\mathbf{A}_j, \mathbf{d}'_j), \forall j \neq i$. We must have $G_i(\mathbf{A}'_i) < G_i(\mathbf{A}_i)$. Otherwise, allocation $\mathbf{A}'$ is preferred over $\mathbf{A}$ by all users and is strictly preferred by user $j \neq i$ w.r.t. the claimed demands $(\mathbf{d}'_{-i}, \mathbf{d}_i)$. This contradicts the Pareto optimality of DRFH allocation. (Recall that allocation $\mathbf{A}$ is an DRFH allocation given the claimed demands $(\mathbf{d}'_{-i}, \mathbf{d}_i)$.) ■

C. Analysis of Important Properties

In addition to the three essential properties shown in the previous subsection, DRFH also provides a number of other important properties. First, since DRFH generalizes DRF to heterogeneous environments, it naturally reduces to the DRF allocation when there is only one server contained in the system, where the global dominant resource defined in DRFH is exactly the same as the dominant resource defined in DRF.

Proposition 4 (Single-server DRF): The DRFH leads to the same allocation as DRF when all resources are concentrated in one server.

Next, by definition, we see that both single-resource fairness and bottleneck fairness trivially hold for the DRFH allocation. We hence omit the proofs of the following two propositions.

Proposition 5 (Single-resource fairness): The DRFH allocation satisfies single-resource fairness.

Proposition 6 (Bottleneck fairness): The DRFH allocation satisfies bottleneck fairness.

Finally, we see that when a user leaves the system and relinquishes all its allocations, the remaining users will not see any reduction of the number of tasks scheduled. Formally,

Proposition 7 (Population monotonicity): The DRFH allocation satisfies population monotonicity.

Proof: Let $\mathbf{A}$ be the resulting DRFH allocation, then for all user i and server l, $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$ and $G_i(\mathbf{A}_i) = g$, where $\{g_{il}\}$ and g solve (6). Suppose user j leaves the system, changing the resulting DRFH allocation to $\mathbf{A}'$. By DRFH, for all user $i \neq j$ and server l, we have $\mathbf{A}'_{il} = g'_{il}\mathbf{d}_i$ and $G_i(\mathbf{A}'_i) = g'$, where $\{g'_{il}\}_{i\neq j}$ and g' solve the following optimization problem:

$$\begin{aligned}\max_{g'_{il}, i\neq j} \quad & g' \\ \text{s.t.} \quad & \textstyle\sum_{i\neq j} g'_{il} d_{ir} \leq c_{lr}, \forall l \in S, r \in R \ , \\ & \textstyle\sum_{l\in U} g'_{il} = g', \forall i \neq j \ .\end{aligned} \tag{7}$$

To show $N_i(\mathbf{A}'_i) \geq N_i(\mathbf{A}_i)$ for all user $i \neq j$, it is equivalent to prove $G_i(\mathbf{A}'_i) \geq G_i(\mathbf{A}_i)$. It is easy to verify that $g, \{g_{il}\}_{i\neq j}$ satisfy all the constraints of (7) and are hence feasible to (7). As a result, $g' \geq g$. This is exactly $G_i(\mathbf{A}'_i) \geq G_i(\mathbf{A}_i)$. ■

D. Discussions of Sharing Incentive

In addition to the aforementioned properties, sharing incentive is another important allocation property that has been frequently mentioned in the literature, *e.g.,* [6], [7], [8], [10], [25]. It ensures that every user's allocation is at least as good as that obtained by evenly partitioning the entire resource pool. When the system contains only a single server, this property is well defined, as evenly dividing the server's resources leads to a *unique* allocation. However, for the system containing multiple heterogeneous servers, there is an *infinite* number of ways to evenly divide the resource pool, and it is unclear which one should be chosen as the benchmark for comparison. For

example, in Fig. 1, two users share the system with 14 CPUs and 14 GB memory in total. The following two allocations both allocate each user 7 CPUs and 7 GB memory: (a) User 1 is allocated 1/2 resources of server 1 and 1/2 resources of server 2, while user 2 receives the rest; (b) user 1 is allocated (1.5 CPUs, 5.5 GB) in server 1 and (5.5 CPUs, 1.5 GB) in server 2, while user 2 receives the rest.

One might think that allocation (a) is a more reasonable benchmark as it allows all n users to have an equal share of every server, each receiving $1/n$ of the server's resources. However, this benchmark has little practical meaning: With a large n, each user will only receive a small fraction of resources on each server, which likely cannot be utilized by any computing task. In other words, having a small slice of resources in each server is essentially meaningless. We therefore consider another benchmark that is more practical.

Since cloud systems are constructed by pooling hundreds of thousands of servers [1], [2], the number of users is typically far smaller than the number of servers [6], [25], *i.e.*, $k \gg n$. An equal division would allocate to each user k/n servers drawn from the same distribution of the system's server configurations. For each user, the allocated k/n servers are then treated as a *dedicated cloud* that is exclusive to the user. The number of tasks scheduled on this dedicated cloud is then used as a benchmark and is compared to the number of tasks scheduled in the original cloud computing system shared with all other users. We will evaluate such a sharing incentive property via trace-driven simulations in Sec. VI.

V. Practical Considerations

So far, all our discussions are based on several assumptions that may not be the case in a real-world system. In this section, we relax these assumptions and discuss how DRFH can be implemented in practice.

A. Weighted Users with a Finite Number of Tasks

In the previous sections, users are assumed to be assigned equal weights and have infinite computing demands. Both assumptions can be easily removed with some minor modifications of DRFH.

When users are assigned uneven weights, let w_i be the weight associated with user i. DRFH seeks an allocation that achieves the *weighted max-min fairness* across users. Specifically, we maximize the minimum *normalized global dominant share* (w.r.t the weight) of all users under the same resource constraints as in (4), *i.e.,*

$$\begin{aligned} \max_{\mathbf{A}} \quad & \min_{i\in U} G_i(\mathbf{A}_i)/w_i \\ \text{s.t.} \quad & \sum_{i\in U} A_{ilr} \le c_{lr}, \forall l \in S, r \in R \ . \end{aligned}$$

When users have a finite number of tasks, the DRFH allocation is computed iteratively. In each round, DRFH increases the global dominant share allocated to all *active users*, until one of them has all its tasks scheduled, after which the user becomes inactive and will no longer be considered in the following allocation rounds. DRFH then starts a new iteration and repeats the allocation process above, until no user is active or no more resources could be allocated to users. Our analysis presented in Sec. IV also extends to weighted users with a finite number of tasks.

B. Scheduling Tasks as Entities

Until now, we have assumed that all tasks are divisible. In a real-world system, however, fractional tasks may not be accepted. To schedule tasks as entities, one can apply *progressive filling* as a simple implementation of DRFH. That is, whenever there is a scheduling opportunity, the scheduler always accommodates the user with the lowest global dominant share. To do this, it picks the first server that fits the user's task. While this First-Fit algorithm offers a fairly good approximation to DRFH, we propose another simple heuristic that can lead to a better allocation with higher resource utilization.

Similar to First-Fit, the heuristic also chooses user i with the lowest global dominant share to serve. However, instead of randomly picking a server, the heuristic chooses the "best" one that most suitably matches user i's tasks, and is hence referred to as the *Best-Fit* DRFH. Specifically, for user i with resource demand vector $\mathbf{D}_i = (D_{i1}, \ldots, D_{im})^T$ and a server l with *available resource vector* $\bar{\mathbf{c}}_l = (\bar{c}_{l1}, \ldots, \bar{c}_{lm})^T$, where $\bar{c}_{lr}$ is the share of resource r remaining available in server l, we define the following *heuristic function* to measure the task's fitness for the server:

$$H(i,l) = \left\| \mathbf{D}_i/D_{i1} - \bar{\mathbf{c}}_l/\bar{c}_{l1} \right\|_1 \ , \tag{8}$$

where $\|\cdot\|_1$ is the L^1-norm. Intuitively, the smaller $H(i,l)$, the more similar the resource demand vector $\mathbf{D}_i$ appears to the server's available resource vector $\bar{\mathbf{c}}_l$, and the better fit user i's task is for server l. For example, a CPU-heavy task is more suitable to run in a server with more available CPU resources. Best-Fit DRFH schedules user i's tasks to server l with the least $H(i,l)$. We evaluate both First-Fit DRFH and Best-Fit DRFH via trace-driven simulations in the next section.

VI. Simulation Results

In this section, we evaluate the performance of DRFH via extensive simulations driven by Google cluster-usage traces [3]. The traces contain resource demand/usage information of over 900 users (*i.e.,* Google services and engineers) on a cluster of 12K servers. The server configurations are summarized in Table I, where the CPUs and memory of each server are normalized so that the maximum server is 1. Each user submits computing jobs, divided into a number of tasks, each requiring a set of resources (*i.e.,* CPU and memory). From the traces, we extract the computing demand information — the required amount of resources and task running time — and use it as the demand input of the allocation algorithms for evaluation.

Dynamic allocation: Our first evaluation focuses on the allocation fairness of the proposed Best-Fit DRFH when users dynamically join and depart the system. We simulate 3 users submitting tasks with different resource requirements to a small cluster of 100 servers. The server configurations are randomly drawn from the distribution of Google cluster servers in Table I, leading to a resource pool containing 52.75 CPU

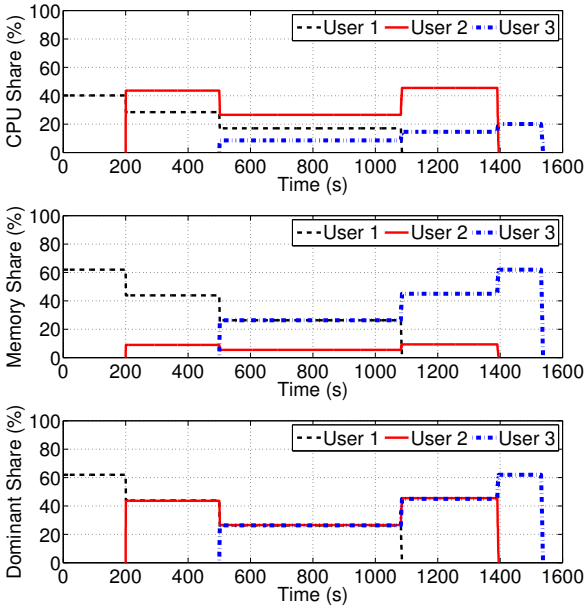


Fig. 4. CPU, memory, and global dominant share for three users on a 100-server system with 52.75 CPU units and 51.32 memory units in total.

TABLE II
RESOURCE UTILIZATION OF THE SLOTS SCHEDULER WITH DIFFERENT SLOT SIZES.

Number of Slots	CPU Utilization	Memory Utilization
10 per maximum server	35.1%	23.4%
12 per maximum server	42.2%	27.4%
14 per maximum server	43.9%	28.0%
16 per maximum server	45.4%	24.2%
20 per maximum server	40.6%	20.0%

units and 51.32 memory units in total. User 1 joins the system at the beginning, requiring 0.2 CPU and 0.3 memory for each of its task. As shown in Fig. 4, since only user 1 is active at the beginning, it is allocated 40% CPU share and 62% memory share. This allocation continues until 200 s, at which time user 2 joins and submits CPU-heavy tasks, each requiring 0.5 CPU and 0.1 memory. Both users now compete for computing resources, leading to a DRFH allocation in which both users receive 44% global dominant share. At 500 s, user 3 starts to submit memory-intensive tasks, each requiring 0.1 CPU and 0.3 memory. The algorithm now allocates the same global dominant share of 26% to all three users until user 1 finishes its tasks and departs at 1080 s. After that, only users 2 and 3 share the system, each receiving the same share on their global dominant resources. A similar process repeats until all users finish their tasks. Throughout the simulation, we see that the Best-Fit DRFH algorithm precisely achieves the DRFH allocation at all times.

Resource utilization: We next evaluate the resource utilization of the proposed Best-Fit DRFH algorithm. We take the 24-hour computing demand data from the Google traces and simulate it on a smaller cloud computing system of 2,000 servers so that fairness becomes relevant. The server configurations are randomly drawn from the distribution of Google cluster servers in Table I. We compare Best-Fit DRFH

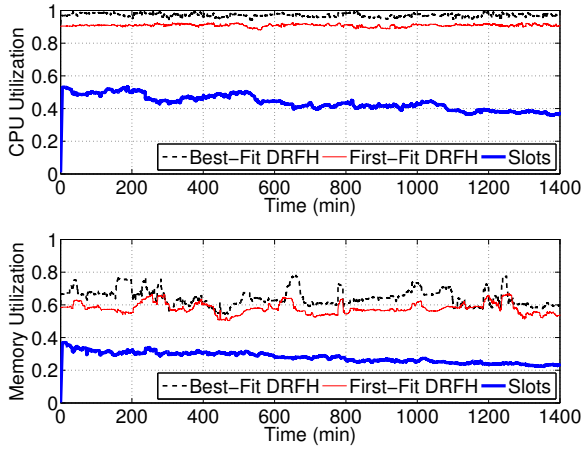
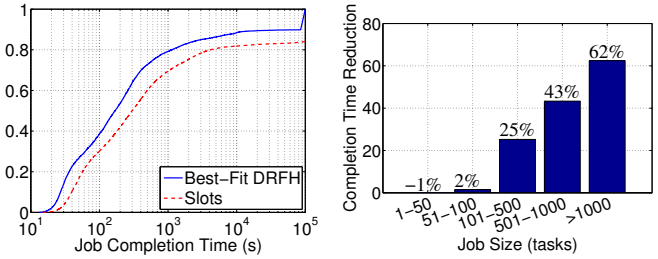


Fig. 5. Time series of CPU and memory utilization.



(a) CDF of job completion times. (b) Job completion time reduction.

Fig. 6. DRFH improvements on job completion times over Slots scheduler.

with two other benchmarks, the traditional Slots schedulers that schedules tasks onto slots of servers (*e.g.,* Hadoop Fair Scheduler [23]), and the First-Fit DRFH that chooses the first server that fits the task. For the former, we try different slot sizes and chooses the one with the highest CPU and memory utilization. Table II summarizes our observations, where dividing the maximum server (1 CPU and 1 memory in Table I) into 14 slots leads to the highest overall utilization.

Fig. 5 depicts the time series of CPU and memory utilization of the three algorithms. We see that the two DRFH implementations significantly outperform the traditional Slots scheduler with much higher resource utilization, mainly because the latter ignores the heterogeneity of both servers and workload. This observation is consistent with findings in the homogeneous environment where all servers are of the same hardware configurations [6]. As for the DRFH implementations, we see that Best-Fit DRFH leads to uniformly higher resource utilization than the First-Fit alternative at all times.

The high resource utilization of Best-Fit DRFH naturally translates to shorter job completion times shown in Fig. 6a, where the CDFs of job completion times for both Best-Fit DRFH and Slots scheduler are depicted. Fig. 6b offers a more detailed breakdown, where jobs are classified into 5 categories based on the number of its computing tasks, and for each category, the mean completion time reduction is computed. While DRFH shows no improvement over Slots scheduler for small jobs, a significant completion time reduction has been observed for those containing more tasks. Generally, the larger the job is, the more improvement one may expect.

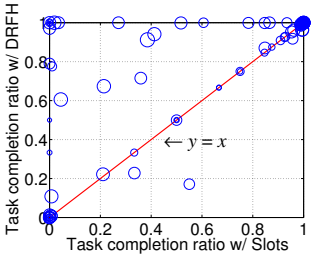


Fig. 7. Task completion ratio of users using Best-Fit DRFH and Slots schedulers, respectively. Each bubble's size is logarithmic to the number of tasks the user submitted.

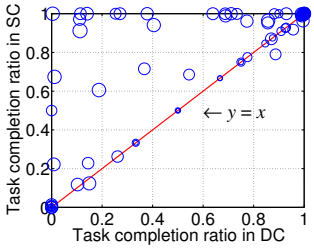


Fig. 8. Task completion ratio of users running on dedicated clouds (DCs) and the shared cloud (SC). Each circle's radius is logarithmic to the number of tasks submitted.

Similar observations have also been found in the homogeneous environments [6].

Fig. 6 does not account for partially completed jobs and focuses only on those having all tasks finished in both Best-Fit and Slots. As a complementary study, Fig. 7 computes the task completion ratio — the number of tasks completed over the number of tasks submitted — for every user using Best-Fit DRFH and Slots schedulers, respectively. The radius of the circle is scaled logarithmically to the number of tasks the user submitted. We see that Best-Fit DRFH leads to higher task completion ratio for almost all users. Around 20% users have all their tasks completed under Best-Fit DRFH but do not under Slots.

Sharing incentive: Our final evaluation is on the sharing incentive property of DRFH. As mentioned in Sec. IV-D, for each user, we run its computing tasks on a dedicated cloud (DC) that is a proportional subset of the original shared cloud (SC). We then compare the task completion ratio in DC with that obtained in SC. Fig. 8 illustrates the results. While DRFH does not guarantee 100% sharing incentive for all users, it benefits most of them by pooling their DCs together. In particular, only 2% users see fewer tasks finished in the shared environment. Even for these users, the task completion ratio decreases only slightly, as can be seen from Fig. 8.

VII. Concluding Remarks

In this paper, we study a multi-resource allocation problem in a heterogeneous cloud computing system where the resource pool is composed of a large number of servers with different configurations in terms of resources such as processing, memory, and storage. The proposed multi-resource allocation mechanism, known as DRFH, equalizes the global dominant share allocated to each user, and hence generalizes the DRF allocation from a single server to multiple heterogeneous servers. We analyze DRFH and show that it retains almost all desirable properties that DRF provides in the single-server scenario. Notably, DRFH is envy-free, Pareto optimal, and truthful. We design a Best-Fit heuristic that implements DRFH in a real-world system. Our large-scale simulations driven by Google cluster traces show that, compared to the traditional single-resource abstraction such as a slot scheduler, DRFH achieves significant improvements in resource utilization, leading to much shorter job completion times.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM SoCC*, 2012.

[3] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google Cluster-Usage Traces," http://code.google.com/p/googleclusterdata/.

[4] "Apache Hadoop," http://hadoop.apache.org.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proc. EuroSys*, 2007.

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

[7] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *Proc. IEEE INFOCOM*, 2012.

[8] D. Dolev, D. Feitelson, J. Halpern, R. Kupferman, and N. Linial, "No justified complaints: On fair sharing of multiple resources," in *Proc. ACM ITCS*, 2012.

[9] A. Gutman and N. Nisan, "Fair allocation without trade," in *Proc. AAMAS*, 2012.

[10] D. Parkes, A. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," in *Proc. ACM EC*, 2012.

[11] S. Baruah, J. Gehrke, and C. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proc. IEEE IPPS*, 1995.

[12] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[13] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, 1998.

[14] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Networking*, vol. 8, no. 5, pp. 556–567, 2000.

[15] J. Kleinberg, Y. Rabani, and É. Tardos, "Fairness in routing and load balancing," in *Proc. IEEE FOCS*, 1999.

[16] J. Blanquer and B. Özden, "Fair queuing for aggregated multiple links," in *Proc. ACM SIGCOMM*, 2001.

[17] Y. Liu and E. Knightly, "Opportunistic fair scheduling over multiple wireless channels," in *Proc. IEEE INFOCOM*, 2003.

[18] C. Koksal, H. Kassab, and H. Balakrishnan, "An analysis of short-term fairness in wireless media access protocols," in *Proc. ACM SIGMETRICS (poster session)*, 2000.

[19] M. Bredel and M. Fidler, "Understanding fairness and its impact on quality of service in IEEE 802.11," in *Proc. IEEE INFOCOM*, 2009.

[20] R. Jain, D. Chiu, and W. Hawe, *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*. Eastern Research Laboratory, Digital Equipment Corporation, 1984.

[21] T. Lan, D. Kao, M. Chiang, and A. Sabharwal, "An axiomatic theory of fairness in network resource allocation," in *Proc. IEEE INFOCOM*, 2010.

[22] "Hadoop Capacity Scheduler," http://hadoop.apache.org/docs/r0.20.2/capacity_scheduler.html.

[23] "Hadoop Fair Scheduler," http://hadoop.apache.org/docs/r0.20.2/fair_scheduler.html.

[24] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. ACM SOSP*, 2009.

[25] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Proc. ACM EuroSys*, 2013.

[26] I. Kash, A. Procaccia, and N. Shah, "No agent left behind: Dynamic fair division of multiple resources," in *Proc. AAMAS*, 2013.

[27] J. Li and J. Xue, "Egalitarian division under Leontief preferences," *Econ. Theory*, vol. 54, no. 3, pp. 597–622, 2013.

[28] A. D. Procaccia, "Cake cutting: Not just child's play," *Commun. ACM*, 2013.

[29] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," University of Toronto, Tech. Rep., 2013. [Online]. Available: http://iqua.ece.toronto.edu/~weiwang/papers/drfh.pdf