Multi-Resource Fair Allocation in Heterogeneous Cloud Computing Systems

Wei Wang, *Student Member, IEEE*, Ben Liang, *Senior Member, IEEE*, and Baochun Li, *Senior Member, IEEE*

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 coalition behavior of misreporting resource demands can benefit all its members. DRFH also ensures some level of service isolation among the users. 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.

Index Terms—Cloud computing, heterogeneous servers, job scheduling, multi-resource allocation, fairness

1 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, cloud computing systems distinguish themselves 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 [2]. 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 [3], [4], [5], [6], [7]. Table 1 illustrates the heterogeneity of servers in one of Google's clusters [3], [8]. Similar server heterogeneity has also been observed in public clouds, such as Amazon EC2 and Rackspace [4], [5].

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 abstraction that falls short. For example, Hadoop [9] and Dryad [10], 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 [11].

Towards addressing the inefficiency of the current allocation module, many recent works focus on *multi-resource allocation* mechanisms. Notably, Ghodsi et al. [11] 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—is equalized. The DRF allocation possesses a set of highly desirable fairness properties, and has quickly received significant attention in the literature [12], [13], [14], [15]. While DRF and its subsequent works address the demand heterogeneity of multiple resources, they all limit the discussion to a simplified model where resources are pooled in one place and the entire resource pool is abstracted as one big server.[1] Such an *all-in-one* resource model not only contrasts the prevalent datacenter infrastructure—where resources are distributed to a large number of servers—but also ignores the server heterogeneity: the allocations depend only on the total amount of resources

• *The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S3G4, Canada. E-mail: weiwang@eceutoronto.ca, liangbli@ece.toronto.edu, bli@eecg.toronto.edu.*

Manuscript received 23 Mar. 2014; revised 18 June 2014; accepted 19 Aug. 2014. Date of publication 8 Oct. 2014; date of current version 2 Sept. 2015. Recommended for acceptance by Y. Liu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2362139

1. While [11] briefly touches on the case where resources are distributed to small servers (known as the discrete scenario), its coverage is rather informal.

TABLE 1
Configurations of Servers in One of Google's Clusters [3], [8]

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

CPU and memory units are normalized to the maximum server (highlighted below).

pooled in the system, irrespective of the underlying resource distribution of servers. In fact, when servers are heterogeneous, even the definition of *dominant resource* is not so clear. Depending on the underlying server configurations, a computing task may bottleneck on different resources in different servers. We shall see that naive extensions, such as applying the DRF allocation to each server separately, may lead to a highly inefficient allocation (details in Section 3.4).

This paper represents a rigorous study to propose a solution with provable operational benefits that bridge the gap between the existing multi-resource allocation models and the state-of-the-art datacenter infrastructure. We propose DRFH, a *DRF* generalization in *Heterogeneous* environments where resources are pooled by a large number 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* resource pool. We systematically analyze DRFH and show that it retains most of the desirable properties that the all-in-one DRF model provides for a single server [11]. 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 one. More importantly, DRFH is *group strategyproof* in that whenever a coalition of users collude with each other to misreport their resource demands, there is a member of the coalition that cannot strictly gain. As a result, the coalition is better off not formed. In addition, DRFH offers some level of service isolation by ensuring the *sharing incentive* property in a weak sense—it allows users to execute more tasks than those under some "equal partition" where the entire resource pool is evenly allocated among all users. DRFH also satisfies a set of other important properties, namely *single-server DRF*, *single-resource fairness*, *bottleneck fairness*, and *population monotonicity* (details in Section 3.3).

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 [8]. Our simulation results show that compared with 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.

The remainder of this paper is organized as follows. We briefly revisit the DRF allocation and point out its limitations in heterogeneous environments in Section 2. We then formulate the allocation problem with heterogeneous servers in Section 3, where a set of desirable allocation properties are also defined. In Section 4, we propose DRFH and analyze its properties. Section 6 dedicates to some practical issues on implementing DRFH. We evaluate the performance of DRFH via trace-driven simulations in Section 6. We survey the related work in Section 7 and conclude the paper in Section 8.

2 Limitations of DRF Allocation in Heterogeneous Systems

In this section, we briefly review the DRF allocation [11] and show that it may lead to an infeasible allocation when a cloud system is composed of multiple heterogeneous servers.

In DRF, the *dominant resource* is defined for each user as the one that requires the largest fraction of the total availability. The mechanism seeks a maximum allocation that equalizes each user's *dominant share*, defined as the fraction of the dominant resource the user has been allocated. Consider an example given in [11]. Suppose that a computing system has nine CPUs and 18 GB memory, and is shared by two users. User 1 wishes to schedule a set of (divisible) tasks each requiring $\langle 1\,\text{CPU},\ 4\,\text{GB}\rangle$, and user 2 has a set of (divisible) tasks each requiring $\langle 3\,\text{CPUs}, 1\,\text{GB}\rangle$. In this example, the dominant resource of user 1 is the memory as each of its task demands 1/9 of the total CPUs and 2/9 of the total memory. On the other hand, the dominant resource of user 2 is CPU, as each of its task requires 1/3 of the total CPUs and 1/18 of the total memory. The DRF mechanism then allocates $\langle 3\,\text{CPUs}, 12\,\text{GB}\rangle$ to user 1 and $\langle 6\,\text{CPUs}, 2\ \text{GB}\rangle$ to user 2, where user 1 schedules three tasks and user 2 schedules two. It is easy to verify that both users receive the same dominant share (i.e., 2/3) and no one can schedule more tasks by allocating additional resources (there is 2 GB memory left unallocated).

The DRF allocation above is based on a simplified all-in-one resource model, where the entire system is modeled as one big server. The allocation hence depends only on the total amount of resources pooled in the system. In the example above, no matter how many servers the system has, and what each server specification is, as long as the system has nine CPUs and 18 GB memory in total, the DRF allocation will always schedule three tasks for user 1 and two for user 2. However, this allocation may not be possible to implement, especially when the system consists of heterogeneous servers. For example, suppose that the resource pool is provided by two servers. Server 1 has one CPU and 14 GB memory, and server 2 has eight CPUs and 4 GB memory. As shown in Fig. 1, even allocating both servers exclusively to user 1, at most two tasks can be scheduled, one in each server. Moreover, even for some server specifications where the DRF allocation is feasible, the mechanism only gives the

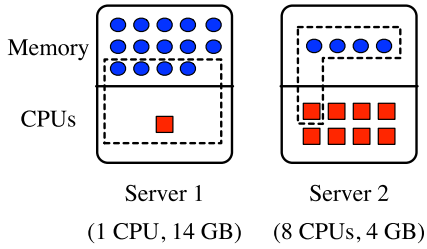


Fig. 1. An example of a system consisting of two heterogeneous servers, in which user 1 can schedule at most two tasks each demanding 1 CPU and 4 GB memory. The resources required to execute the two tasks are also highlighted in the figure.

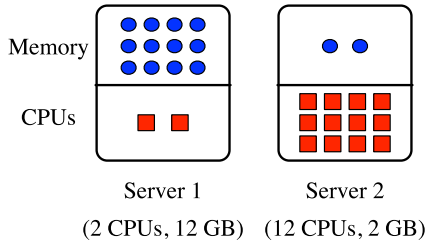


Fig. 2. An example of a system containing two heterogeneous servers shared by two users. Each computing task of user 1 requires 0.2 CPU time and 1 GB memory, while the computing task of user 2 requires 1 CPU time and 0.2 GB memory.

total amount of resources each user should receive. It remains unclear how many resources a user should be allocated in *each server*. These problems significantly limit the application of the DRF mechanism. In general, the allocation is valid only when the system contains a single server or multiple homogeneous servers, which is rarely a case under the prevalent datacenter infrastructure.

Despite the limitation of the all-in-one resource model, DRF is shown to possess a set of highly desirable allocation properties for cloud computing systems [11], [15]. A natural question is: how should the DRF intuition be generalized to a heterogeneous environment to achieve similar properties? Note that this is not an easy question to answer. In fact, with heterogeneous servers, even the definition of dominant resource is not so clear. Depending on the server specifications, a resource most demanded in one server (in terms of the fraction of the server's availability) might be the least-demanded in another. For instance, in the example of Fig. 1, user 1 demands CPU the most in server 1. But in server 2, it demands memory the most. Should the dominant resource be defined separately for each server, or should it be defined for the entire resource pool? How should the allocation be conducted? And what properties do the resulting allocation preserve? We shall answer these questions in the following sections.

3 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.

3.1 Basic Settings

Let $S = \{1, \ldots, k\}$ be the set of heterogeneous servers a cloud computing system has in its resource pool. Let $R = \{1, \ldots, m\}$ be the set of m hardware resources provided by each server, e.g., CPU, memory, storage, etc. Let $\mathbf{c}_l = (c_{l1}, \ldots, c_{lm})^T$ be the *resource capacity vector* of server $l \in S$, where each component c_{lr} denotes the total amount of resource r available in this server. Without loss of generality, we normalize the total availability of every resource to 1, i.e.,

$$\sum_{l \in S} c_{lr} = 1, \quad \forall r \in R.$$

Let $U = \{1, \ldots, n\}$ be the set of cloud users sharing the entire 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 amount of resource r required by each instance of the task of user i. For simplicity, we assume positive demands, i.e., $D_{ir} > 0$ for all user i and resource r. We say resource r_i^* the *global dominant resource* of user i if

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

In other words, resource r_i^* is the most heavily demanded resource required by each instance of the task of user i, over the entire resource pool. For each 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. 2 where the system consists of two heterogeneous servers. Server 1 is high-memory with two CPUs and 12 GB memory, while server 2 is high-CPU with 12 CPUs and 2 GB memory. Since the system has 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 that 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 [11], [13], [14], [15], [16]. We shall discuss how these assumptions can be relaxed in Section 5.

3.2 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 amount 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 $\mathbf{A}$ *feasible* if no server is required to use more than any of its total resources, i.e.,

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

For each user i, given allocation $\mathbf{A}_{il}$ in server l, let $N_{il}(\mathbf{A}_{il})$ be the maximum number of tasks (possibly fractional) it can schedule. We have

$$N_{il}(\mathbf{A}_{il}) D_{ir} \leq A_{ilr}, \quad \forall r \in R.$$

As a result,

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

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 [17], we call such an allocation *non-wasteful*:

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

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

Further, 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. Unless otherwise specified, we limit the discussion to non-wasteful allocations.

3.3 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 [11], [12], [18] and the economics literature [17], [19].

Envy-freeness. An allocation mechanism is *envy-free* if no user prefers the other's allocation to its own, i.e.,

$$N_i(\mathbf{A}_i) \geq N_i(\mathbf{A}_j), \quad \forall i, j \in U.$$

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 $j \neq i$ such that $N_j(\mathbf{A}'_j) < N_j(\mathbf{A}_j)$. In other words, allocation $\mathbf{A}$ cannot be further improved such that 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 to achieve high resource utilization.

Group strategyproofness. An allocation mechanism is *group strategyproof* if whenever a coalition of users misreport their resource demands (assuming a user's demand is its private information), there is a member of the coalition who would schedule less tasks and hence has no incentive to join the coalition. Specifically, let $M \subset U$ be the coalition of manipulators in which user $i \in M$ misreports its demand as $\mathbf{D}'_i \neq \mathbf{D}_i$. Let $\mathbf{A}'$ be the allocation returned. Also, let $\mathbf{A}$ be the allocation returned when all users truthfully report their demands. The allocation mechanism is group strategyproof if there exists a manipulator $i \in M$ who cannot schedule more tasks than being truthful, i.e.,

$$N_i(\mathbf{A}'_i) \leq N_i(\mathbf{A}_i).$$

In other words, user i is better off quiting the coalition. Group strategyproofness is of a special importance for a cloud computing system, as it is common to observe in a real-world system that users try to manipulate the scheduler for more allocations by lying about their resource demands [11], [18].

Sharing incentive is another critical property that has been frequently mentioned in the literature [11], [12], [13], [15]. 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 selected as a benchmark to compare with. We shall give a specific discussion to Section 4.5, where we justify between two reasonable alternatives.

In addition to the four essential allocation properties above, we also consider four other important properties as follows:

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.

To summarize, our objective is to design an allocation mechanism that guarantees all the properties defined above.

3.4 Naive DRF Extension and Its Inefficiency

It has been shown in [11], [15] that the DRF allocation satisfies all the desirable properties mentioned above when the entire resource pool is modeled as one server. When resources are distributed to multiple heterogeneous servers, a naive generalization is to separately apply the DRF allocation per server. For instance, consider the example of Fig. 2.

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

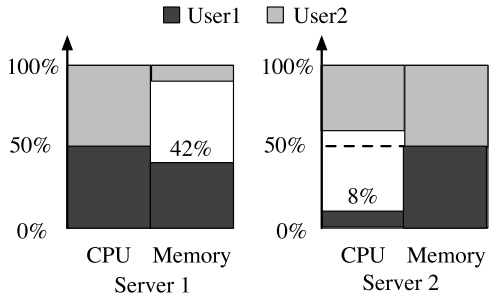


Fig. 3. DRF allocation for the example shown in Fig. 2, where user 1 is allocated five tasks in server 1 and 1 in server 2, while user 2 is allocated one task in server 1 and 5 in server 2.

We first apply DRF in server 1. Because CPU is the dominant resource of both users, it is equally divided for both of them, each receiving 1. As a result, user 1 schedules five tasks onto server 1, while user 2 schedules one. Similarly, in server 2, memory is the dominant resource of both users and is evenly allocated, leading to one task scheduled for user 1 and five for user 2. The resulting allocations in the two servers are illustrated in Fig. 3, where both users schedule six tasks.

Unfortunately, this allocation violates the 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, almost twice the number of tasks scheduled under the DRF allocation. In fact, a similar example can be constructed to show that the per-server DRF may lead to *arbitrarily low* resource utilization. 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.

4 DRFH Allocation and Its Properties

In this section, we present 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 Section 3.

4.1 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

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

be the amount of global dominant resource user i is allocated in server l. Since the total availability of resources is normalized to 1, we also refer to $G_{il}(\mathbf{A}_{il})$ the *global dominant share* user i receives in server l. Simply adding up $G_{il}(\mathbf{A}_{il})$ over all servers gives the global dominant share user i receives under allocation $\mathbf{A}_i$, i.e.,

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

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 Section 3.2). We have the following structural result. Its proof is deferred to the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2362139.[3]

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.,

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

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 every 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

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

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

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

Note that the second constraint embodies the fairness in terms of 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. The allocation received by each user i in server l is simply $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$.

For example, Fig. 4 illustrates the resulting DRFH allocation in the example of Fig. 2. By solving (6), DRFH allocates

3. The appendix is given in a supplementary document as per the TPDS submission guidelines.

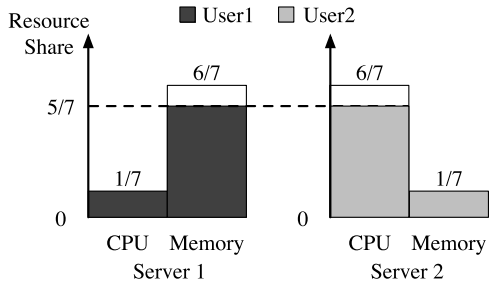


Fig. 4. An alternative allocation with higher system utilization for the example of Fig. 2. Server 1 and 2 are exclusively assigned to user 1 and 2, respectively. Both users schedule 10 tasks.

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). Our analyses of DRFH start with the four essential resource allocation properties, namely, envy-freeness, Pareto optimality, group strategyproofness, and sharing incentive.

4.2 Envy-Freeness

We first show by the following proposition 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 $\{g_{il}\}, g$ be the solution to problem (6). For all user i, its DRFH allocation in server l is $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$. 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

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

where the inequality holds because

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

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

4.3 Pareto Optimality

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 $\{g_{il}\}, g$ 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, $\{g_{il}\}$ is also the solution to (5), and g is 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 $G_i(\mathbf{A}'_i) \ge G_i(\mathbf{A}_i)$ 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 $\{g_{il}\}, g$ optimally solve (5).

To see this, consider user j. We have

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

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 to increase their global dominant share, therefore strictly improving the objective of (5).

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 to 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

$$\begin{aligned} \sum_i \hat{g}_{il_0}d_{ir} &= \sum_i \left(g''_{il_0} + \delta\right)d_{ir} \\ &= \sum_i g'_{il_0}d_{ir} - \epsilon d_{jr} + \delta\sum_i d_{ir} \\ &\le c_{l_0r} - \left(\epsilon d_{jr} - \delta\sum_i d_{ir}\right) \\ &\le c_{l_0r}. \end{aligned}$$

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

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

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

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

4.4 Group Strategyproofness

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 [11], [18]. 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 dominant strategy for all users, even if they form a coalition to misreport together with others.

Proposition 3 (Group strategyproofness). *The DRFH allocation obtained by solving (6) is group strategyproof in that the coalition behavior of misreporting demands cannot strictly benefit every member.*

Proof. Let $M \subset U$ be the set of strategic users forming a coalition to misreport the normalized demand vector $\mathbf{d}'_M = (\mathbf{d}'_i)_{i \in M}$, where $\mathbf{d}'_i \neq \mathbf{d}_i$. for all $i \in M$. Let $\mathbf{d}'$ be the collection of normalized demand vectors submitted by all users, where $\mathbf{d}'_i = \mathbf{d}_i$, for all $i \in U \setminus M$. Let $\mathbf{A}'$ be the resulting allocation obtained by solving (6). In particular, $\mathbf{A}'_{il} = g'_{il}\mathbf{d}'_i$ for each user i and server l, and $g' = \sum_l g'_{il}$, where $\{g'_{il}\}, g'$ solve (6). On the other hand, let $\mathbf{A}$ be the allocation returned when all users truthfully report their demands, and $\{g_{il}\}, g$ the solution to (6) with the truthful $\mathbf{d}$. Similarly, for each user i and server l, we have $\mathbf{A}_{il} = g_{il}\mathbf{d}_i$, and $g = \sum_l g_{il}$. We check the following two cases and show that there exists a user $i \in M$, such 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 all user $i \in M$. Clearly,

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

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

$$\begin{aligned} G_i(\mathbf{A}'_i) &= \sum_l G_{il}(\mathbf{A}'_{il}) \\ &= \sum_l G_{il}(g'_{il}\mathbf{d}'_i) \\ &= \sum_l \min_r\{g'_{il}d'_{ir}/d_{ir}\} \\ &= \rho_i g' \\ &\leq g \\ &= G_i(\mathbf{A}_i). \end{aligned}$$

Case 2: $g' > g$. We first consider users that are not manipulators. Since they truthfully report their demands, we have

$$G_j(\mathbf{A}'_j) = g' > g = G_j(\mathbf{A}_j), \forall j \in U \setminus M. \quad (7)$$

Now for those manipulators, there is a user $i \in M$ such that $G_i(\mathbf{A}'_i) < G_i(\mathbf{A}_i)$. Otherwise, allocation $\mathbf{A}'$ is strictly preferred to allocation $\mathbf{A}$ by all users. This contradicts the facts that $\mathbf{A}$ is a Pareto optimal allocation and $\mathbf{A}'$ is a feasible allocation. □

4.5 Sharing Incentive

In addition to the aforementioned three properties, sharing incentive is another critical allocation property that has been frequently mentioned in the literature, e.g., [11], [12], [13], [15], [18]. The property ensures that every user can execute at least the number of tasks it schedules when the entire resource pool is evenly partitioned. The property provides service isolations among the users.

While the sharing incentive property is well defined in the all-in-one resource model, it is not for the system with multiple heterogeneous servers. In the former case, since the entire resource pool is abstracted as a single server, evenly dividing every resource of this big server would lead to a *unique* allocation. However, when the system consists of multiple heterogeneous servers, there are many different ways to evenly divide these servers, and it is unclear which one should be used as a benchmark for comparison. For instance, in the example of Fig. 2, two users share a system with 14 CPUs and 14 GB memory in total. The following two allocations both allocate each user seven 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 is allocated 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 is allocated the rest. It is easy to verify that the two allocations lead to different number of tasks scheduled for the same user, and can be used as two different allocation benchmarks. In fact, one can construct many other allocations that evenly divide all resources among the users.

Despite the general ambiguity explained above, in the next two sections, we consider two definitions of the sharing incentive property, strong and weak, depending on the choice of the benchmark for equal partitioning of resources.

4.5.1 Strong Sharing Incentive

Among various allocations that evenly divide all servers, perhaps the most straightforward approach is to evenly partition each server's availability $\mathbf{c}_l$ among all n users. The strong sharing incentive property is defined by using this per-server partitioning as a benchmark.

Definition 2 (Strong sharing incentive). *Allocation* $\mathbf{A}$ *satisfies the* strong sharing incentive *property if each user schedules fewer tasks by evenly partitioning each server, i.e.,*

$$N_i(\mathbf{A}_i) = \sum_{l \in S} N_i(\mathbf{A}_{il}) \geq \sum_{l \in S} N_i(\mathbf{c}_l/n), \quad \forall i \in U.$$

Before we proceed, it is worth mentioning that the per-server partitioning above cannot be directly implemented in practice. With a large number of users, in each server, everyone will be allocated a very small fraction of the server's availability. In practice, such a small slice of resources

usually cannot be used to run any computing task. However, per-server partitioning may be interpreted as follows. Since a cloud system is constructed by pooling hundreds of thousands of servers [2], [3], the number of users is typically far smaller than the number of servers [11], [18], i.e., $k \gg n$. An equal partition could randomly allocate to each user k/n servers, which is equivalent to randomly allocating each server to each user with probability $1/n$. It is easy to see that the *mean number* of tasks scheduled for each user under this random allocation is $\sum_l N_i(\mathbf{c}_i/n)$, the same as that obtained under the per-server partitioning.

Unfortunately, the following proposition shows that DRFH may violate the sharing incentive property in the strong sense. The proof gives a counterexample.

Proposition 4. *DRFH does not satisfy the property of strong sharing incentive.*

Proof. Consider a system consisting of two servers. Server 1 has 1 CPU and 2 GB memory; server 2 has four CPUs and 3 GB memory. There are two users. Each instance of the task of user 1 demands 1 CPU and 1 GB memory; each of user 2's tasks demands three CPUs and 2 GB memory. In this case, we have $\mathbf{c}_1 = (1/5, 2/5)^T$, $\mathbf{c}_2 = (4/5, 3/5)^T$, $\mathbf{D}_1 = (1/5, 1/5)^T$, $\mathbf{D}_2 = (3/5, 2/5)^T$, $\mathbf{d}_1 = (1, 1)^T$, and $\mathbf{d}_2 = (1, 2/3)^T$. It is easy to verify that under DRFH, the global dominant share both users receive is $12/25$. On the other hand, under the per-server partitioning, the global dominant share that user 2 receives is $1/2$, higher than that received under DRFH. □

While DRFH may violate the strong sharing incentive property, we shall show via trace-driven simulations in Section 6 that this only happens in rare cases.

4.5.2 Weak Sharing Incentive

The strong sharing incentive property is defined by choosing the per-server partitioning as a benchmark, which is only one of many different ways to evenly divide the total availability. In general, any equal partition that allocates an equal share of every resource can be used as a benchmark. This allows us to relax the sharing incentive definition. We first define an equal partition as follows.

Definition 3 (Equal partition). *Allocation* $\mathbf{A}$ *is an* equal partition *if it divides every resource evenly among all users, i.e.,*

$$\sum_{l \in S} A_{ilr} = 1/n, \quad \forall r \in R, i \in U.$$

It is easy to verify that the aforementioned per-server partition is an equal partition. We are now ready to define the weak sharing incentive property as follows.

Definition 4 (Weak sharing incentive). *Allocation* $\mathbf{A}$ *satisfies the* weak sharing incentive *property if there exists an equal partition* $\mathbf{A}'$ *under which each user schedules fewer tasks than those under* $\mathbf{A}$*, i.e.,*

$$N_i(\mathbf{A}_i) \geq N_i(\mathbf{A}'_i), \quad \forall i \in U.$$

In other words, the property of weak sharing incentive only requires the allocation to be better off than one equal partition, without specifying its specific form. It is hence a more relaxed requirement than the strong sharing incentive property.

The following proposition shows that DRFH satisfies the sharing incentive property in the weak sense. The proof is constructive.

Proposition 5 (Weak sharing incentive). *DRFH satisfies the property of weak sharing incentive.*

Proof. Let g be the global dominant share each user receives under a DRFH allocation $\mathbf{A}$, and g_{il} the global dominant share user i receives in server l. We construct an equal partition $\mathbf{A}'$ under which users schedule fewer tasks than those under $\mathbf{A}$.

Case 1: $g \geq 1/n$. In this case, let $\mathbf{A}'$ be any equal partition. We show that each user schedules fewer tasks under $\mathbf{A}'$ than those under $\mathbf{A}$. To see this, consider the DRFH allocation $\mathbf{A}$. Since it is non-wasteful, the number of tasks user i schedules is

$$N_i(\mathbf{A}_i) = g/D_{ir_i^*} \geq 1/nD_{ir_i^*}.$$

On the other hand, the number of tasks user i schedules under $\mathbf{A}'$ would be at most

$$\begin{aligned} N_i(\mathbf{A}'_i) &= \sum_{l \in S} \min_r \{A'_{ilr}/D_{ir}\} \\ &\leq \sum_{l \in S} A'_{ilr}/D_{ir_i^*} \\ &= 1/nD_{ir_i^*} \\ &\leq N_i(\mathbf{A}_i). \end{aligned}$$

Case 2: $g < 1/n$. In this case, no resource has been fully allocated under $\mathbf{A}$, i.e.,

$$\sum_{i \in U} \sum_{l \in S} A_{ilr} = \sum_{i \in U} \sum_{l \in S} g_{il} d_{ir} \leq \sum_{i \in U} \sum_{l \in S} g_{il} = \sum_{i \in U} g < 1$$

for all resource $r \in R$. Let

$$L_{lr} = c_{lr} - \sum_{i \in U} A_{ilr}$$

be the amount of resource r left unallocated in server l, Further, let

$$L_r = \sum_{l \in S} L_{lr} = 1 - \sum_{i \in U} \sum_{l \in S} A_{ilr}$$

be the total amount of resource r left unallocated.

We are now ready to construct an equal partition $\mathbf{A}'$ based on $\mathbf{A}$. Since $\mathbf{A}'$ should allocate each user $1/n$ of the total availability of every resource r, the additional amount of resource r user i needs to obtain is

$$u_{ir} = 1/n - \sum_{l \in S} A_{ilr}.$$

It is easy to see that $u_{ir} > 0, \forall i \in U, r \in R$. The demanded fraction of unallocated resource r for user i is

$$f_{ir} = u_{ir}/L_r.$$

As a result, we can construct $\mathbf{A}'$ by reallocating those left-over resources in each server to users, in proportion to their demands, i.e.,

$$A'_{ilr} = A_{ilr} + L_{lr} f_{ir}, \quad \forall i \in U, l \in S, r \in R.$$

It is easy to verify that $\mathbf{A}'$ is an equal partition, i.e.,

$$\begin{aligned}\sum_{l\in S} A'_{ilr} &= \sum_{l\in S} L_{lr} f_{ir} + \sum_{l\in S} A_{ilr} \\ &= (u_{ir}/L_r) \sum_{l\in S} L_{lr} + \sum_{l\in S} A_{ilr} \\ &= u_{ir} + \sum_{l\in S} A_{ilr} \\ &= 1/n, \qquad \forall i \in U, r \in R.\end{aligned}$$

We now compare the number of tasks scheduled for each user under both allocations $\mathbf{A}$ and $\mathbf{A}'$. Because $\mathbf{A}'$ allocates more resources to each user than $\mathbf{A}$ does, we have $N_i(\mathbf{A}'_i) \geq N_i(\mathbf{A}_i)$ for all i. On the other hand, by the Pareto optimality of allocation $\mathbf{A}$, no user can schedule more tasks without decreasing the number of tasks scheduled for others. Therefore, we must have $N_i(\mathbf{A}'_i) = N_i(\mathbf{A}_i)$ for all i. □

4.5.3 Discussion

Strong sharing incentive provides more *predictable* service isolation than weak sharing incentive does. It assures a user *a priori* that it can schedule at least the number of tasks when every server is evenly allocated. This gives users a concrete idea on the worst Quality of Service (QoS) they may receive, allowing them to accurately predict their computing performance. While weak sharing incentive also provides some degree of service isolation, a user cannot infer the guaranteed number of tasks it can schedule *a priori* from this weaker property, and therefore cannot predict the computing performance.

We note that the root cause of such degradation of service isolation is due to the heterogeneity among servers. When all servers are of the same specification of hardware resources, DRFH reduces to DRF, and strong sharing incentive is guaranteed. This is also the case for schedulers adopting the single-resource abstraction. For example, in Hadoop, each server is divided into several slots (e.g., reducers and mappers). Hadoop Fair Scheduler [20] allocates these slots evenly to all users. We see that predictable service isolation is achieved: each user receives at least k_s/n slots, where k_s is the number of slots, and n is the number of users.

In general, one can view weak sharing incentive as the price paid by DRFH to achieve high resource utilization. In fact, naively applying DRF allocation separately to each server retains strong sharing incentive: in each server, the DRF allocation ensures that a user can schedule at least the number of tasks when resources are evenly allocated [11], [15]. However, as we have seen in Section 3.4, such a naive DRF extension may lead to extremely low resource utilization that is unacceptable. Similar problem also exists for traditional schedulers adopting single-resource abstractions. By artificially dividing servers into slots, these schedulers cannot match computing demands to available resources at a fine granularity, resulting in poor resource utilization in practice [11]. For these reasons, we believe that slightly trading off the degree of service isolation for much higher resource utilization is well justified. We shall use trace-driven simulation to show in Section 6.3 that DRFH only violates strong sharing incentive in rare cases in the Google cluster.

4.6 Other Important Properties

In addition to the four essential properties shown in the previous section, 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 6 (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 7 (Single-resource fairness). *The DRFH allocation satisfies single-resource fairness.*

Proposition 8 (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 9 (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 that 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:

$$\begin{aligned}\max_{g'_{il}, i\neq j} \quad & g' \\ \text{s.t.} \quad & \sum_{i\neq j} g'_{il} d_{ir} \leq c_{lr}, \forall l \in S, r \in R, \\ & \sum_{l\in U} g'_{il} = g', \forall i \neq j.\end{aligned} \tag{8}$$

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 (8) and are hence feasible to (8). As a result, $g' \geq g$, which is exactly $G_i(\mathbf{A}'_i) \geq G_i(\mathbf{A}_i)$. □

5 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.

5.1 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. Because each iteration saturates at least one user's resource demand, the allocation will be accomplished in at most n rounds, where n is the number of users.[4] Our analysis presented in Section 4 also extends to weighted users with a finite number of tasks.

5.2 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.[5] 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 whose remaining resources are sufficient to accommodate the request of 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 whose remaining resoruces most suitably matches the demand of 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* $\overline{\mathbf{c}}_l = (\overline{c}_{l1}, \ldots, \overline{c}_{lm})^T$, where $\overline{c}_{lr}$ is the share of resource r remaining available in server l, we define the following *heuristic function* to quantitatively measure the fitness of the task for server l:

$$H(i,l) = \|\mathbf{D}_i/D_{i1} - \overline{\mathbf{c}}_l/\overline{c}_{l1}\|_1, \tag{9}$$

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 $\overline{\mathbf{c}}_l$, and the better fit user i's task is for server l. Best-Fit DRFH schedules user i's tasks to server l with the least $H(i,l)$.

As an illustrative example, suppose that only two types of resources are concerned, CPU and memory. A CPU-heavy task of user i with resource demand vector $\mathbf{D}_i = (1/10, 1/30)^T$ is to be scheduled, meaning that the task requires $1/10$ of the total CPU availability and $1/30$ of the total memory availability of the system. Only two servers have sufficient remaining resources to accommodate this task. Server 1 has the available resource vector $\overline{\mathbf{c}}_1 = (1/5, 1/15)^T$; Server 2 has the available resource vector $\overline{\mathbf{c}}_2 = (1/8, 1/4)^T$. Intuitively, because the task is CPU-bound, it is more fit for Server 1, which is CPU-abundant. This is indeed the case as $H(i,1) = 0 < H(i,2) = 5/3$, and Best-Fit DRFH places the task onto Server 1.

Both First-Fit and Best-Fit DRFH can be easily implemented by searching all k servers in $O(k)$ time, which is fast enough for small- and medium-sized clusters. For a large cluster containing tens of thousands of servers, this computation can be fast approximated by adapting the power of two choices load balancing technique [21]. Instead of scanning through all servers, the scheduler randomly probes two servers and places the task on the server that fits the task better.

It is worth mentioning that the definition of the heuristic function (9) is not unique. In fact, one can use more complex heuristic function other than (9) to measure the fitness of a task for a server, e.g., *cosine similarity* [22]. However, as we shall show in the next section, Best-Fit DRFH with (9) as its heuristic function already improves the utilization to a level where the system capacity is almost saturated. Therefore, the benefit of using more complex fitness measure is very limited, at least for the Google cluster traces [8].

6 Trace-Driven Simulation

In this section, we evaluate the performance of DRFH via extensive simulations driven by Google cluster-usage traces [8]. 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 1, 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.

6.1 Dynamic Allocation

Our first evaluation focuses on the allocation fairness of the proposed Best-Fit DRFH when users dynamically join and

4. For medium- and large-sized cloud clusters, n is in the order of thousands [3], [8].

5. Progressive filling has also been used to implement the DRF allocation [11]. However, when the system consists of multiple heterogeneous servers, progressive filling will lead to a DRFH allocation.

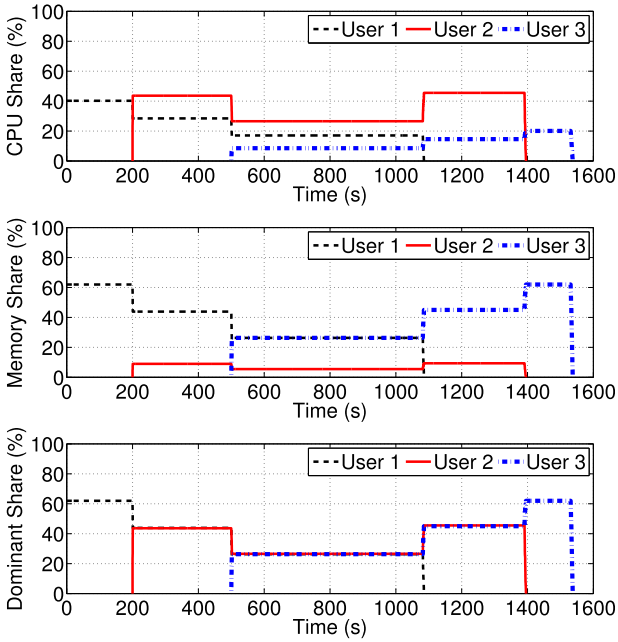


Fig. 5. CPU, memory, and global dominant share for three users on a 100-server system with 52.75 CPU units and 51.32 memory units in total.

depart the system. We simulate three 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 1, leading to a resource pool containing 52.75 CPU 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. 5, since only user 1 is active at the beginning, it is allocated 40 percent CPU share and 62 percent 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 percent 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 percent 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.

TABLE 2
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%

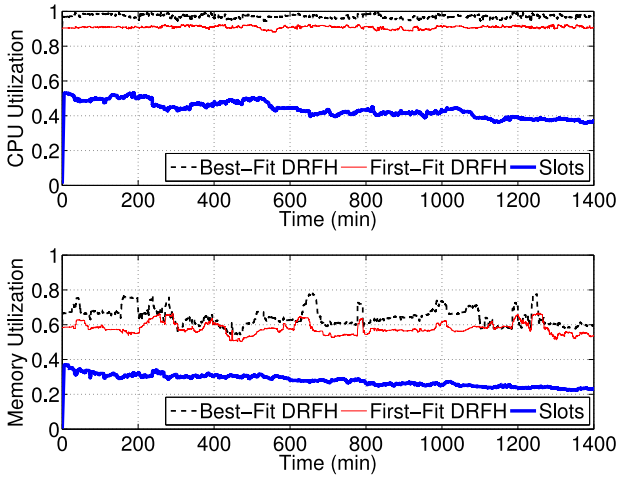


Fig. 6. Time series of CPU and memory utilization.

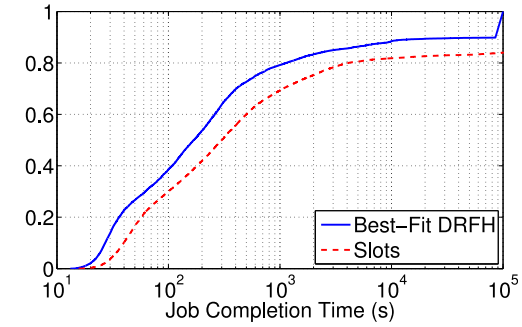
6.2 Resource Utilization

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 1. We compare Best-Fit DRFH with two other benchmarks, the traditional Slots schedulers that schedules tasks onto slots of servers (e.g., Hadoop Fair Scheduler [20]), 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 2 summarizes our observations, where dividing the maximum server (1 CPU and 1 memory in Table 1) into 14 slots leads to the highest overall utilization.

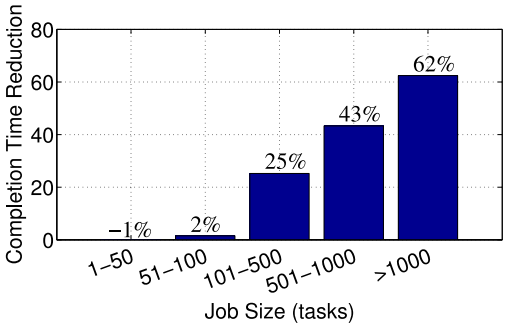
Fig. 6 depicts the time series of CPU and memory utilization of the three algorithms. We see that the two DRFH implementations significantly outperform the traditional Slots scheduler with much higher resource utilization, mainly because the latter ignores the heterogeneity of both servers and workload. This observation is consistent with findings in the homogeneous environment where all servers are of the same hardware configurations [11]. 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. 7a, where the CDFs of job completion times for both Best-Fit DRFH and Slots scheduler are depicted. Fig. 7b offers a more detailed breakdown, where jobs are classified into five 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. Similar observations have also been found in the homogeneous environments [11].

Fig. 7 does not account for partially completed jobs and focuses only on those having all tasks finished in



(a) CDF of job completion times.



(b) Job completion time reduction.

Fig. 7. DRFH improvements on job completion times over Slots scheduler.

both Best-Fit and Slots. As a complementary study, Fig. 8 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 percent users have all their tasks completed under Best-Fit DRFH but do not under Slots.

6.3 Sharing Incentive

Our final evaluation is on the sharing incentive property of DRFH. While we have shown in Section 4.5 that DRFH may not satisfy the property in the strong sense, it remains unclear how often this property would be violated in practice. Is it frequently happened or just a rare case? We answer this question in this section.

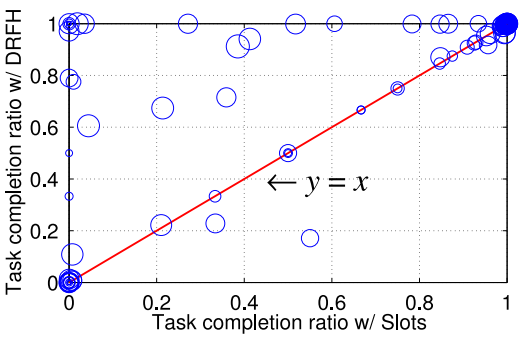


Fig. 8. Task completion ratio of users using Best-Fit DRFH and Slots schedulers, respectively. Each bubble's size is logarithmic to the number of tasks the user submitted.

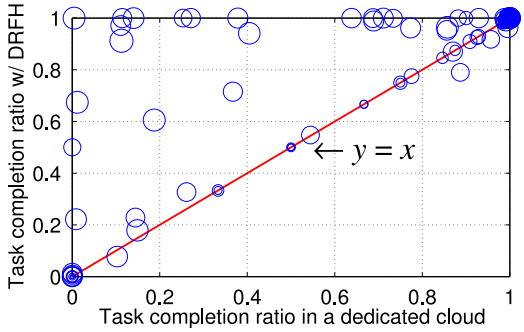


Fig. 9. Comparison of task completion ratios under DRFH and that obtained in dedicated clouds (DCs). Each circle's radius is logarithmic to the number of tasks submitted.

We first compute the task completion ratio under the benchmark per-server partition. To do this, we randomly choose $\lceil k/n \rceil$ servers, where k is the number of servers and n is the number of users in the traces. We then allocate these $\lceil k/n \rceil$ servers to a user and schedule its tasks onto them. These $\lceil k/n \rceil$ servers form a *dedicated cloud* exclusive for this user. We compute the task completion ratio obtained in this dedicated cloud and compare it with the one obtained under the DRFH allocation. Fig. 9 illustrates the comparison results for all users. We see that most users would prefer DRFH allocation as compared with running tasks in a dedicated cloud. In particular, only 2 percent users see fewer tasks finished under the DRFH allocation. Even for these users, the task completion ratio decreases only slightly, as shown in Fig. 9. As a result, we see that DRFH only violates the property of strong sharing incentive in rare cases in the Google traces.

7 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 [23], [24] and link bandwidth [25], [26], [27], [28], [29]. Various fairness notions have also been proposed throughout the years, ranging from application-specific allocations [30], [31] to general fairness measures [25], [32], [33].

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 [20], [34] partition a node into slots with fixed fractions of resources, and allocate resources jointly at the slot granularity. Quincy [35], a fair scheduler developed for Dryad [10], models the fair scheduling problem as a min-cost flow problem to schedule jobs into slots. The recent work [18] takes the job placement constraints into consideration, yet it still uses a slot-based single resource abstraction.

Ghodsi et al. [11] are the first in the literature to present a systematic investigation on the multi-resource allocation problem in cloud computing systems. They proposed 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. [12] generalized the DRF measure and incorporated it into a unifying

framework that captures the trade-offs between allocation fairness and efficiency. Dolev et al. [13] suggested 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 [14] considered another settings of DRF with a more general domain of user utilities, and showed their connections to the BBF mechanism. Parkes et al. [15], on the other hand, extended 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 studied the loss of social welfare under the DRF rules. Kash et al. [16] extended the DRF model to allow users to join the system over time but will never leave. Bhattacharya et al. [36] generalized DRF to a hierarchical scheduler that offers service isolations in a computing system with a hierarchical structure. All these works assume, explicitly or implicitly, that resources are either concentrated into one super computer, or are distributed to a set of homogeneous servers with exactly the same resource configuration.

However, server heterogeneity has been widely observed in today's cloud computing systems. Specifically, Ahmad et al. [6] noted that datacenter clusters usually consist of both high-performance servers and low-power nodes with different hardware architectures. Reiss et al. [3], [8] illustrated a wide range of server specification in Google clusters. As for public clouds, Farley et al. [4] and Ou et al. [5] observed significant hardware diversity among Amazon EC2 servers that may lead to substantially different performance across supposedly equivalent VM instances. Ou et al. [5] also pointed out that such server heterogeneity is not limited to EC2 only, but generally exists in long-lasting public clouds such as Rackspace.

To our knowledge, the very recent paper [37] is the only work that studied allocation properties in the presence of server heterogeneity, where a randomized allocation algorithm, called Constrained-DRF (CDRF), is proposed to schedule *discrete jobs*. While CDRF possess all the desirable properties discussed in this paper, it is too complex for a job scheduler, and an efficient algorithm remains an open problem [37]. More recently, Grandl et al. [38] proposed an efficient heuristic algorithm for multi-resource scheduling in heterogeneous computer clusters. Their work mainly focuses on designing a good heuristic algorithm, not studying the allocation properties, and is therefore orthogonal to our work.

Other related works include fair-division problems in the economics literature, in particular the *egalitarian division* under Leontief preferences [17] and the *cake-cutting problem* [19]. These works also assume the *all-in-one* resource model, and hence cannot be directly applied to cloud computing systems with heterogeneous servers.

8 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 group strategyproof. It also offers the sharing incentive in a weak sense. 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.

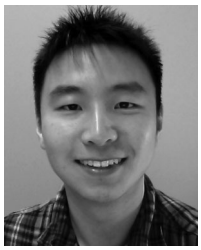
Acknowledgments

Part of this paper has appeared in [1]. This new version contains substantial revision with new illustrative examples, property analyses, proofs, and discussions.

References

[1] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 583–591.
[2] 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.
[3] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM 3rd Symp. Cloud Comput.*, 2012, p. 7.
[4] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: Exploiting performance heterogeneity in public clouds," in *Proc. ACM 3rd Symp. Cloud Comput.*, 2012, p. 20.
[5] Z. Ou, H. Zhuang, A. Lukyanenko, J. Nurminen, P. Hui, V. Mazalov, and A. Yla-Jaaski, "Is the same instance type created equal? Exploiting heterogeneity of public clouds," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 201–214, Jul.–Dec. 2013.
[6] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," in *Proc. 17th Int. Conf. Archit. Support Programm. Lang. Oper. Syst.*, 2012, pp. 61–74.
[7] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting platform heterogeneity for power efficient data centers," in *Proc. USENIX 4th Int. Conf Auton. Comput.*, 2007, p. 5.
[8] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google Cluster-Usage Traces. (2012). [Online]. Available: http://code.google.com/p/googleclusterdata/
[9] Apache Hadoop. (2014). [Online]. Available: http://hadoop.apache.org
[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM Eur. Conf. Comput. Syst.*, 2007, pp. 59–72.
[11] 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 Conf. Netw. Syst. Des. Implementation*, 2011, pp. 323–336.
[12] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *Proc. IEEE Conf. Comput. Commun.*, 2012, pp. 1206–1214.
[13] D. Dolev, D. Feitelson, J. Halpern, R. Kupferman, and N. Linial, "No justified complaints: On fair sharing of multiple resources," in *Proc. 3rd Innovations Theor. Comput. Sci. Conf.*, 2012, pp. 68–75.
[14] A. Gutman, and N. Nisan, "Fair allocation without trade," in *Proc. 11th Int. Conf. Auton. Agents Multiagent Syst.*, 2012, pp. 719–728.
[15] D. Parkes, A. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," in *Proc. 13th ACM Conf. Electron. Commerce*, 2012, pp. 808–825.

[16] I. Kash, A. Procaccia, and N. Shah, "No agent left behind: Dynamic fair division of multiple resources," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2013, pp. 351–358.
[17] J. Li and J. Xue, "Egalitarian division under Leontief preferences," *Econ. Theory*, vol. 54, no. 3, pp. 597–622, 2013.
[18] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Proc. ACM 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 365–378.
[19] A. D. Procaccia, "Cake cutting: Not just child's play," *Commun. ACM*, vol. 56, no. 7, pp. 78–87, 2013.
[20] Hadoop Fair Scheduler. (2013). [Online]. Available: http://hadoop.apache.org/docs/r0.20.2/fair_scheduler.html
[21] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.
[22] A. Singhal, "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, Dec. 2001.
[23] S. Baruah, J. Gehrke, and C. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proc. IEEE 9th Int. Parallel Process. Symp.*, 1995, pp. 280–288.
[24] 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.
[25] 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.
[26] J. Mo, and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, Oct. 2000.
[27] J. Kleinberg, Y. Rabani, and É. Tardos, "Fairness in routing and load balancing," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 1999, pp. 568.
[28] J. Blanquer and B. Özden, "Fair queuing for aggregated multiple links," in *Proc. ACM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2001, pp. 189–197.
[29] Y. Liu and E. Knightly, "Opportunistic fair scheduling over multiple wireless channels," in *Proc. IEEE Conf. Comput. Commun.*, 2003, pp. 1106–1115.
[30] C. Koksal, H. Kassab, and H. Balakrishnan, "An analysis of short-term fairness in wireless media access protocols," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2000, pp. 118–119.
[31] M. Bredel and M. Fidler, "Understanding fairness and its impact on quality of service in IEEE 802.11," in *Proc. IEEE Conf. Comput. Commun.*, 2009, pp. 1098–1106.
[32] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer system," Eastern Res. Laboratory, Digital Equipment Corporation, Maynard, MA, USA, Tech. Rep. DEC-TR-301, 1984.
[33] T. Lan, D. Kao, M. Chiang, and A. Sabharwal, "An axiomatic theory of fairness in network resource allocation," in *Proc. IEEE Conf. Comput. Commun.*, 2010, pp. 1–9.
[34] Hadoop Capacity Scheduler. (2013). [Online]. Available: http://hadoop.apache.org/docs/r0.20.2/capacity_scheduler.html
[35] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. ACM 22nd Symp. Oper. Syst. Principle*, 2009, pp. 261–276.
[36] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical scheduling for diverse datacenter workloads," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, p. 4.
[37] E. Friedman, A. Ghodsi, and C.-A. Psomas, "Strategyproof allocation of discrete jobs on multiple machines," in *Proc. 15th ACM Conf. Econ. Comput.*, 2014, pp. 529–546.
[38] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. ACM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2014, pp. 455–466.



Wei Wang received the BEngr and MASc degrees from the Department of Electrical Engineering, Shanghai Jiao Tong University, in 2007 and 2010. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering at the University of Toronto. His general research interests cover the broad area of computer networking, with special emphasis on resource management and scheduling in cloud computing systems. He is also interested in problems at the intersection of computer networking and economics. He is a student member of the IEEE.

Ben Liang received the honors-simultaneous BSc (valedictorian) and MSc degrees in electrical engineering from Polytechnic University in Brooklyn, NY, in 1997, and the PhD degree in electrical engineering with computer science minor from Cornell University in Ithaca, New York, in 2001. In the 2001-2002 academic year, he was a visiting lecturer and post-doctoral research associate at Cornell University. He joined the Department of Electrical and Computer Engineering at the University of Toronto in 2002, where he is currently a professor. His current research interests are in mobile communications and networked systems. He is an editor for the *IEEE Transactions on Wireless Communications* and an associate editor for the *Wiley Security and Communication Networks journal*, in addition to regularly serving on the organizational or technical committee of a number of conferences. He is a senior member of the IEEE and a member of ACM and Tau Beta Pi.

Baochun Li received the BEngr degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995, and the MS and PhD degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a professor. He holds the Nortel Networks Junior chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed chair in computer engineering since August 2005. His research interests include large-scale multimedia systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He received the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he received the Multimedia Communications Best Paper Award from the IEEE Communications Society, and the University of Toronto McLean Award. He is a member of the ACM and a senior member of the IEEE.