

Dominant Resource Fairness: Fair Allocation of Multiple Resource Types

Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, Ion Stoica
University of California, Berkeley

{alig,matei,benh,andyk,shenker,istoica}@cs.berkeley.edu

Abstract

We consider the problem of fair resource allocation in a system containing different resource types, where each user may have different demands for each resource. To address this problem, we propose *Dominant Resource Fairness (DRF)*, a generalization of max-min fairness to multiple resource types. We show that DRF, unlike other possible policies, satisfies several highly desirable properties. First, DRF incentivizes users to share resources, by ensuring that no user is better off if resources are equally partitioned among them. Second, DRF is strategy-proof, as a user cannot increase her allocation by lying about her requirements. Third, DRF is envy-free, as no user would want to trade her allocation with that of another user. Finally, DRF allocations are Pareto efficient, as it is not possible to improve the allocation of a user without decreasing the allocation of another user. We have implemented DRF in the Mesos cluster resource manager, and show that it leads to better throughput and fairness than the slot-based fair sharing schemes in current cluster schedulers.

1 Introduction

Resource allocation is a key building block of any shared computer system. One of the most popular allocation policies proposed so far has been *max-min fairness*, which maximizes the minimum allocation received by a user in the system. Assuming each user has enough demand, this policy gives each user an equal share of the resources. Max-min fairness has been generalized to include the concept of weight, where each user receives a share of the resources proportional to its weight.

The attractiveness of weighted max-min fairness stems from its generality and its ability to provide performance isolation. The weighted max-min fairness model can support a variety of other resource allocation policies, including priority, reservation, and deadline based allocation [31]. In addition, weighted max-min fairness ensures isolation, in that a user is guaranteed to receive

her share irrespective of the demand of the other users.

Given these features, it should come as no surprise that a large number of algorithms have been proposed to implement (weighted) max-min fairness with various degrees of accuracy, such as round-robin, proportional resource sharing [32], and weighted fair queueing [12]. These algorithms have been applied to a variety of resources, including link bandwidth [8, 12, 15, 24, 27, 29], CPU [11, 28, 31], memory [4, 31], and storage [5].

Despite the vast amount of work on fair allocation, the focus has so far been primarily on a *single* resource type. Even in multi-resource environments, where users have heterogeneous resource demands, allocation is typically done using a single resource abstraction. For example, fair schedulers for Hadoop and Dryad [1, 18, 34], two widely used cluster computing frameworks, allocate resources at the level of fixed-size partitions of the nodes, called *slots*. This is despite the fact that different jobs in these clusters can have widely different demands for CPU, memory, and I/O resources.

In this paper, we address the problem of fair allocation of multiple types of resources to users with heterogeneous demands. In particular, we propose Dominant Resource Fairness (DRF), a generalization of max-min fairness for multiple resources. The intuition behind DRF is that in a multi-resource environment, the allocation of a user should be determined by the user's *dominant share*, which is the maximum share that the user has been allocated of any resource. In a nutshell, DRF seeks to maximize the minimum dominant share across all users. For example, if user *A* runs CPU-heavy tasks and user *B* runs memory-heavy tasks, DRF attempts to equalize user *A*'s share of CPUs with user *B*'s share of memory. In the single resource case, DRF reduces to max-min fairness for that resource.

The strength of DRF lies in the properties it satisfies. These properties are trivially satisfied by max-min fairness for a single resource, but are non-trivial in the case of multiple resources. Four such properties are

sharing incentive, strategy-proofness, Pareto efficiency, and envy-freeness. DRF provides incentives for users to share resources by guaranteeing that no user is better off in a system in which resources are statically and equally partitioned among users. Furthermore, DRF is strategy-proof, as a user cannot get a better allocation by lying about her resource demands. DRF is Pareto-efficient as it allocates all available resources subject to satisfying the other properties, and without preempting existing allocations. Finally, DRF is envy-free, as no user prefers the allocation of another user. Other solutions violate at least one of the above properties. For example, the preferred [3, 22, 33] fair division mechanism in microeconomic theory, Competitive Equilibrium from Equal Incomes [30], is not strategy-proof.

We have implemented and evaluated DRF in Mesos [16], a resource manager over which multiple cluster computing frameworks, such as Hadoop and MPI, can run. We compare DRF with the slot-based fair sharing scheme used in Hadoop and Dryad and show that slot-based fair sharing can lead to poorer performance, unfairly punishing certain workloads, while providing weaker isolation guarantees.

While this paper focuses on resource allocation in datacenters, we believe that DRF is generally applicable to other multi-resource environments where users have heterogeneous demands, such as in multi-core machines.

The rest of this paper is organized as follows. Section 2 motivates the problem of multi-resource fairness. Section 3 lists fairness properties that we will consider in this paper. Section 4 introduces DRF. Section 5 presents alternative notions of fairness, while Section 6 analyzes the properties of DRF and other policies. Section 7 provides experimental results based on traces from a Facebook Hadoop cluster. We survey related work in Section 8 and conclude in Section 9.

2 Motivation

While previous work on weighted max-min fairness has focused on single resources, the advent of cloud computing and multi-core processors has increased the need for allocation policies for environments with multiple resources and heterogeneous user demands. By *multiple resources* we mean resources of different *types*, instead of multiple instances of the same interchangeable resource.

To motivate the need for multi-resource allocation, we plot the resource usage profiles of tasks in a 2000-node Hadoop cluster at Facebook over one month (October 2010) in Figure 1. The placement of a circle in Figure 1 indicates the memory and CPU resources consumed by tasks. The size of a circle is logarithmic to the number of tasks in the region of the circle. Though the majority of tasks are CPU-heavy, there exist tasks that are memory-

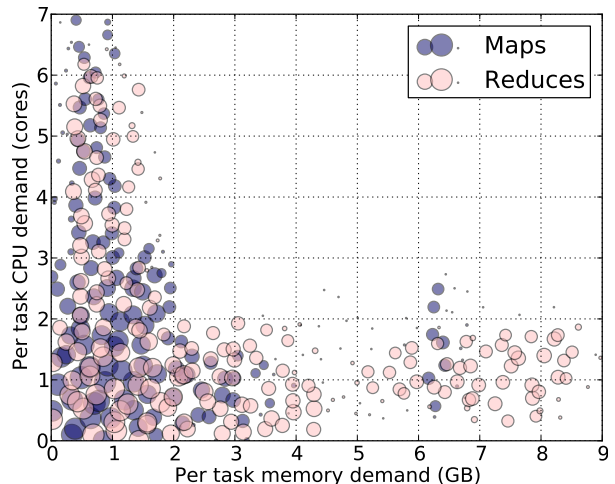


Figure 1: CPU and memory demands of tasks in a 2000-node Hadoop cluster at Facebook over one month (October 2010). Each bubble’s size is logarithmic in the number of tasks in its region.

heavy as well, especially for reduce operations.

Existing fair schedulers for clusters, such as Quincy [18] and the Hadoop Fair Scheduler [2, 34], ignore the heterogeneity of user demands, and allocate resources at the granularity of slots, where a slot is a fixed fraction of a node. This leads to inefficient allocation as a slot is more often than not a poor match for the task demands.

Figure 2 quantifies the level of fairness and isolation provided by the Hadoop MapReduce fair scheduler [2, 34]. The figure shows the CDFs of the ratio between the task CPU demand and the slot CPU share, and of the ratio between the task memory demand and the slot memory share. We compute the slot memory and CPU shares by simply dividing the total amount of memory and CPUs by the number of slots. A ratio of 1 corresponds to a perfect match between the task demands and slot resources, a ratio below 1 corresponds to tasks underutilizing their slot resources, and a ratio above 1 corresponds to tasks over-utilizing their slot resources, which may lead to thrashing. Figure 2 shows that most of the tasks either underutilize or overutilize some of their slot resources. Modifying the number of slots per machine will not solve the problem as this may result either in a lower overall utilization or more tasks experiencing poor performance due to over-utilization (see Section 7).

3 Allocation Properties

We now turn our attention to designing a max-min fair allocation policy for multiple resources and heterogeneous requests. To illustrate the problem, consider a system consisting of 9 CPUs and 18 GB RAM, and two users: user *A* runs tasks that require $\langle 1 \text{ CPUs}, 4 \text{ GB} \rangle$ each, and user *B* runs tasks that require $\langle 3 \text{ CPUs}, 1 \text{ GB} \rangle$ each. What constitutes a fair allocation policy for this case?

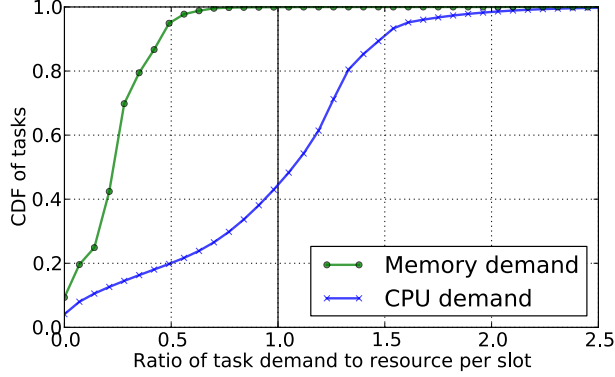


Figure 2: CDF of demand to slot ratio in a 2000-node cluster at Facebook over a one month period (October 2010). A demand to slot ratio of 2.0 represents a task that requires twice as much CPU (or memory) than the slot CPU (or memory) size.

One possibility would be to allocate each user half of every resource. Another possibility would be to equalize the aggregate (*i.e.*, CPU plus memory) allocations of each user. While it is relatively easy to come up with a variety of possible “fair” allocations, it is unclear how to evaluate and compare these allocations.

To address this challenge, we start with a set of desirable properties that we believe any resource allocation policy for multiple resources and heterogeneous demands should satisfy. We then let these properties guide the development of a fair allocation policy. We have found the following four properties to be important:

1. *Sharing incentive*: Each user should be better off sharing the cluster, than exclusively using her own partition of the cluster. Consider a cluster with identical nodes and n users. Then a user should not be able to allocate more tasks in a cluster partition consisting of $\frac{1}{n}$ of all resources.
2. *Strategy-proofness*: Users should not be able to benefit by lying about their resource demands. This provides incentive compatibility, as a user cannot improve her allocation by lying.
3. *Envy-freeness*: A user should not prefer the allocation of another user. This property embodies the notion of fairness [13, 30].
4. *Pareto efficiency*: It should not be possible to increase the allocation of a user without decreasing the allocation of at least another user. This property is important as it leads to maximizing system utilization subject to satisfying the other properties.

We briefly comment on the strategy-proofness and sharing incentive properties, which we believe are of special importance in datacenter environments. Anecdotal evidence from cloud operators that we have talked

with indicates that strategy-proofness is important, as it is common for users to attempt to manipulate schedulers. For example, one of Yahoo!’s Hadoop MapReduce datacenters has different numbers of slots for map and reduce tasks. A user discovered that the map slots were contended, and therefore launched all his jobs as long reduce phases, which would manually do the work that MapReduce does in its map phase. Another big search company provided dedicated machines for jobs only if the users could guarantee high utilization. The company soon found that users would sprinkle their code with infinite loops to artificially inflate utilization levels.

Furthermore, any policy that satisfies the sharing incentive property also provides performance isolation, as it guarantees a minimum allocation to each user (*i.e.*, a user cannot do worse than owning $\frac{1}{n}$ of the cluster) irrespective of the demands of the other users.

It can be easily shown that in the case of a single resource, max-min fairness satisfies all the above properties. However, achieving these properties in the case of multiple resources and heterogeneous user demands is not trivial. For example, the preferred fair division mechanism in microeconomic theory, Competitive Equilibrium from Equal Incomes [22, 30, 33], is not strategy-proof (see Section 6.1.2).

In addition to the above properties, we consider four other nice-to-have properties:

- *Single resource fairness*: For a single resource, the solution should reduce to max-min fairness.
- *Bottleneck fairness*: If there is one resource that is percent-wise demanded most of by *every* user, then the solution should reduce to max-min fairness for that resource.
- *Population monotonicity*: When a user leaves the system and relinquishes her resources, none of the allocations of the remaining users should decrease.
- *Resource monotonicity*: If more resources are added to the system, none of the allocations of the existing users should decrease.

4 Dominant Resource Fairness (DRF)

We propose Dominant Resource Fairness (DRF), a new allocation policy for multiple resources that meets all four of the required properties in the previous section. For every user, DRF computes the share of each resource allocated to that user. The maximum among all shares of a user is called that user’s *dominant share*, and the resource corresponding to the dominant share is called the *dominant resource*. Different users may have different dominant resources. For example, the dominant resource of a user running a computation-bound job is

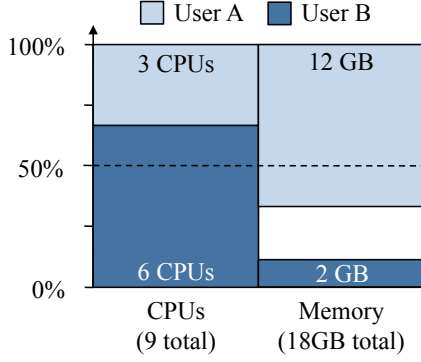


Figure 3: DRF allocation for the example in Section 4.1.

CPU, while the dominant resource of a user running an I/O-bound job is bandwidth.¹ DRF simply applies max-min fairness across users' dominant shares. That is, DRF seeks to maximize the smallest dominant share in the system, then the second-smallest, and so on.

We start by illustrating DRF with an example (§4.1), then present an algorithm for DRF (§4.2) and a definition of weighted DRF (§4.3). In Section 5, we present two other allocation policies: asset fairness, a straightforward policy that aims to equalize the aggregate resources allocated to each user, and competitive equilibrium from equal incomes (CEEI), a popular fair allocation policy preferred in the micro-economic domain [22, 30, 33].

In this section, we consider a computation model with n users and m resources. Each user runs individual tasks, and each task is characterized by a *demand vector*, which specifies the amount of resources required by the task, e.g., $\langle 1 \text{ CPU}, 4 \text{ GB} \rangle$. In general, tasks (even the ones belonging to the same user) may have different demands.

4.1 An Example

Consider a system with 9 CPUs, 18 GB RAM, and two users, where user A runs tasks with demand vector $\langle 1 \text{ CPU}, 4 \text{ GB} \rangle$, and user B runs tasks with demand vector $\langle 3 \text{ CPUs}, 1 \text{ GB} \rangle$ each.

In the above scenario, each task from user A consumes $1/9$ of the total CPUs and $2/9$ of the total memory, so user A 's dominant resource is memory. Each task from user B consumes $1/3$ of the total CPUs and $1/18$ of the total memory, so user B 's dominant resource is CPU. DRF will equalize users' dominant shares, giving the allocation in Figure 3: three tasks for user A , with a total of $\langle 3 \text{ CPUs}, 12 \text{ GB} \rangle$, and two tasks for user B , with a total of $\langle 6 \text{ CPUs}, 2 \text{ GB} \rangle$. With this allocation, each user ends up with the *same* dominant share, i.e., user A gets $2/3$ of RAM, while user B gets $2/3$ of the CPUs.

This allocation can be computed mathematically as follows. Let x and y be the number of tasks allocated

¹A user may have the same share on multiple resources, and might therefore have multiple dominant resources.

Algorithm 1 DRF pseudo-code

$R = \langle r_1, \dots, r_m \rangle$ \triangleright total resource capacities
 $C = \langle c_1, \dots, c_m \rangle$ \triangleright consumed resources, initially 0
 s_i ($i = 1..n$) \triangleright user i 's dominant shares, initially 0
 $U_i = \langle u_{i,1}, \dots, u_{i,m} \rangle$ ($i = 1..n$) \triangleright resources given to user i , initially 0

pick user i with lowest dominant share s_i
 $D_i \leftarrow$ demand of user i 's next task
if $C + D_i \leq R$ **then**
 $C = C + D_i$ \triangleright update consumed vector
 $U_i = U_i + D_i$ \triangleright update i 's allocation vector
 $s_i = \max_{j=1}^m \{u_{i,j}/r_j\}$
else
return \triangleright the cluster is full
end if

by DRF to users A and B , respectively. Then user A receives $\langle x \text{ CPU}, 4x \text{ GB} \rangle$, while user B gets $\langle 3y \text{ CPU}, y \text{ GB} \rangle$. The total amount of resources allocated to both users is $(x + 3y)$ CPUs and $(4x + y)$ GB. Also, the dominant shares of users A and B are $4x/18 = 2x/9$ and $3y/9 = y/3$, respectively (their corresponding shares of memory and CPU). The DRF allocation is then given by the solution to the following optimization problem:

$$\begin{aligned}
 &\max (x, y) && \text{(Maximize allocations)} \\
 &\text{subject to} \\
 &x + 3y \leq 9 && \text{(CPU constraint)} \\
 &4x + y \leq 18 && \text{(Memory constraint)} \\
 &\frac{2x}{9} = \frac{y}{3} && \text{(Equalize dominant shares)}
 \end{aligned}$$

Solving this problem yields² $x = 3$ and $y = 2$. Thus, user A gets $\langle 3 \text{ CPU}, 12 \text{ GB} \rangle$ and B gets $\langle 6 \text{ CPU}, 2 \text{ GB} \rangle$.

Note that DRF need not always equalize users' dominant shares. When a user's total demand is met, that user will not need more tasks, so the excess resources will be split among the other users, much like in max-min fairness. In addition, if a resource gets exhausted, users that do not need that resource can still continue receiving higher shares of the other resources. We present an algorithm for DRF allocation in the next section.

4.2 DRF Scheduling Algorithm

Algorithm 1 shows pseudo-code for DRF scheduling. The algorithm tracks the total resources allocated to each user as well as the user's dominant share, s_i . At each step, DRF picks the user with the lowest dominant share among those with tasks ready to run. If that user's task demand can be satisfied, i.e., there are enough resources

²Note that given last constraint (i.e., $2x/9 = y/3$) allocations x and y are simultaneously maximized.

Schedule	User <i>A</i>		User <i>B</i>		CPU total alloc.	RAM total alloc.
	res. shares	dom. share	res. shares	dom. share		
User <i>B</i>	$\langle 0, 0 \rangle$	0	$\langle 3/9, 1/18 \rangle$	1/3	3/9	1/18
User <i>A</i>	$\langle 1/9, 4/18 \rangle$	2/9	$\langle 3/9, 1/18 \rangle$	1/3	4/9	5/18
User <i>A</i>	$\langle 2/9, 8/18 \rangle$	4/9	$\langle 3/9, 1/18 \rangle$	1/3	5/9	9/18
User <i>B</i>	$\langle 2/9, 8/18 \rangle$	4/9	$\langle 6/9, 2/18 \rangle$	2/3	8/9	10/18
User <i>A</i>	$\langle 3/9, 12/18 \rangle$	2/3	$\langle 6/9, 2/18 \rangle$	2/3	1	14/18

Table 1: Example of DRF allocating resources in a system with 9 CPUs and 18 GB RAM to two users running tasks that require $\langle 1 \text{ CPU}, 4 \text{ GB} \rangle$ and $\langle 3 \text{ CPUs}, 1 \text{ GB} \rangle$, respectively. Each row corresponds to DRF making a scheduling decision. A row shows the shares of each user for each resource, the user’s dominant share, and the fraction of each resource allocated so far. DRF repeatedly selects the user with the lowest dominant share (indicated in bold) to launch a task, until no more tasks can be allocated.

available in the system, one of her tasks is launched. We consider the general case in which a user can have tasks with *different* demand vectors, and we use variable D_i to denote the demand vector of the next task user i wants to launch. For simplicity, the pseudo-code does not capture the event of a task finishing. In this case, the user releases the task’s resources and DRF again selects the user with the smallest dominant share to run her task.

Consider the two-user example in Section 4.1. Table 1 illustrates the DRF allocation process for this example. DRF first picks B to run a task. As a result, the shares of B become $\langle 3/9, 1/18 \rangle$, and the dominant share becomes $\max(3/9, 1/18) = 1/3$. Next, DRF picks A , as her dominant share is 0. The process continues until it is no longer possible to run new tasks. In this case, this happens as soon as CPU has been saturated.

At the end of the above allocation, user A gets $\langle 3 \text{ CPU}, 12 \text{ GB} \rangle$, while user B gets $\langle 6 \text{ CPU}, 2 \text{ GB} \rangle$, *i.e.*, each user gets $2/3$ of its dominant resource.

Note that in this example the allocation stops as soon as any resource is saturated. However, in the general case, it may be possible to continue to allocate tasks even after some resource has been saturated, as some tasks might not have any demand on the saturated resource.

The above algorithm can be implemented using a binary heap that stores each user’s dominant share. Each scheduling decision then takes $O(\log n)$ time for n users.

4.3 Weighted DRF

In practice, there are many cases in which allocating resources equally across users is not the desirable policy. Instead, we may want to allocate more resources to users running more important jobs, or to users that have contributed more resources to the cluster. To achieve this goal, we propose Weighted DRF, a generalization of both DRF and weighted max-min fairness.

With Weighted DRF, each user i is associated a weight vector $W_i = \langle w_{i,1}, \dots, w_{i,m} \rangle$, where $w_{i,j}$ represents the weight of user i for resource j . The definition of a dominant share for user i changes to $s_i = \max_j \{u_{i,j}/w_{i,j}\}$, where $u_{i,j}$ is user i ’s share of resource j . A particular

case of interest is when all the weights of user i are equal, *i.e.*, $w_{i,j} = w_i$, ($1 \leq j \leq m$). In this case, the ratio between the dominant shares of users i and j will be simply w_i/w_j . If the weights of all users are set to 1, Weighted DRF reduces trivially to DRF.

5 Alternative Fair Allocation Policies

Defining a fair allocation in a multi-resource system is not an easy question, as the notion of “fairness” is itself open to discussion. In our efforts, we considered numerous allocation policies before settling on DRF as the only one that satisfies all four of the required properties in Section 3: sharing incentive, strategy-proofness, Pareto efficiency, and envy-freeness. In this section, we consider two of the alternatives we have investigated: Asset Fairness, a simple and intuitive policy that aims to equalize the aggregate resources allocated to each user, and Competitive Equilibrium from Equal Incomes (CEEI), the policy of choice for fairly allocating resources in the microeconomic domain [22, 30, 33]. We compare these policies with DRF in Section 5.3.

5.1 Asset Fairness

The idea behind Asset Fairness is that equal shares of different resources are worth the same, *i.e.*, that 1% of all CPUs worth is the same as 1% of memory and 1% of I/O bandwidth. Asset Fairness then tries to equalize the aggregate resource value allocated to each user. In particular, Asset Fairness computes for each user i the aggregate share $x_i = \sum_j s_{i,j}$, where $s_{i,j}$ is the share of resource j given to user i . It then applies max-min across users’ aggregate shares, *i.e.*, it repeatedly launches tasks for the user with the minimum aggregate share.

Consider the example in Section 4.1. Since there are twice as many GB of RAM as CPUs (*i.e.*, 9 CPUs and 18 GB RAM), one CPU is worth twice as much as one GB of RAM. Supposing that one GB is worth \$1 and one CPU is worth \$2, it follows that user A spends \$6 for each task, while user B spends \$7. Let x and y be the number of tasks allocated by Asset Fairness to users A and B , respectively. Then the asset-fair allocation is

given by the solution to the following optimization problem:

$$\begin{aligned}
& \max (x, y) && \text{(Maximize allocations)} \\
& \text{subject to} \\
& x + 3y \leq 9 && \text{(CPU constraint)} \\
& 4x + y \leq 18 && \text{(Memory constraint)} \\
& 6x = 7y && \text{(Every user spends the same)}
\end{aligned}$$

Solving the above problem yields $x = 2.52$ and $y = 2.16$. Thus, user A gets $\langle 2.5 \text{ CPUs}, 10.1 \text{ GB} \rangle$, while user B gets $\langle 6.5 \text{ CPUs}, 2.2 \text{ GB} \rangle$, respectively.

While this allocation policy seems compelling in its simplicity, it has a significant drawback: it *violates* the sharing incentive property. As we show in Section 6.1.1, asset fairness can result in one user getting less than $1/n$ of all resources, where n is the total number of users.

5.2 Competitive Equilibrium from Equal Incomes

In microeconomic theory, the preferred method to fairly divide resources is Competitive Equilibrium from Equal Incomes (CEEI) [22, 30, 33]. With CEEI, each user receives initially $\frac{1}{n}$ of every resource, and subsequently, each user trades her resources with other users in a perfectly competitive market.³ The outcome of CEEI is both envy-free and Pareto efficient [30].

More precisely, the CEEI allocation is given by the *Nash bargaining solution*⁴ [22, 23]. The Nash bargaining solution picks the feasible allocation that maximizes $\prod_i u_i(a_i)$, where $u_i(a_i)$ is the utility that user i gets from her allocation a_i . To simplify the comparison, we assume that the utility that a user gets from her allocation is simply her dominant share, s_i .

Consider again the two-user example in Section 4.1. Recall that the dominant share of user A is $4x/18 = 2x/9$ while the dominant share of user B is $3y/9 = y/3$, where x is the number of tasks given to A and y is the number of tasks given to B . Maximizing the product of the dominant shares is equivalent to maximizing the product $x \cdot y$. Thus, CEEI aims to solve the following optimization problem:

$$\begin{aligned}
& \max (x \cdot y) && \text{(maximize Nash product)} \\
& \text{subject to} \\
& x + 3y \leq 9 && \text{(CPU constraint)} \\
& 4x + y \leq 18 && \text{(Memory constraint)}
\end{aligned}$$

Solving the above problem yields $x = 45/11$ and $y = 18/11$. Thus, user A gets $\langle 4.1 \text{ CPUs}, 16.4 \text{ GB} \rangle$, while user B gets $\langle 4.9 \text{ CPUs}, 1.6 \text{ GB} \rangle$.

³A perfect market satisfies the price-taking (*i.e.*, no single user affects prices) and market-clearance (*i.e.*, matching supply and demand via price adjustment) assumptions.

⁴For this to hold, utilities have to be homogeneous, *i.e.*, $u(\alpha x) = \alpha u(x)$ for $\alpha > 0$, which is true in our case.

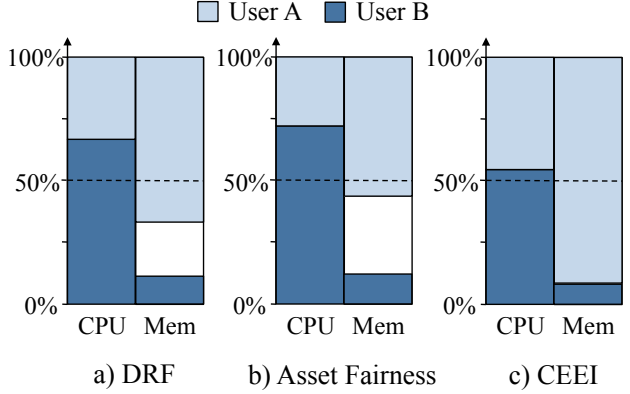


Figure 4: Allocations given by DRF, Asset Fairness and CEEI in the example scenario in Section 4.1.

Unfortunately, while CEEI is envy-free and Pareto efficient, it turns out that it is *not* strategy-proof, as we will show in Section 6.1.2. Thus, users can increase their allocations by lying about their resource demands.

5.3 Comparison with DRF

To give the reader an intuitive understanding of Asset Fairness and CEEI, we compare their allocations for the example in Section 4.1 to that of DRF in Figure 4.

We see that DRF equalizes the dominant shares of the users, *i.e.*, user A 's memory share and user B 's CPU share. In contrast, Asset Fairness equalizes the *total fraction of resources* allocated to each user, *i.e.*, the areas of the rectangles for each user in the figure. Finally, because CEEI assumes a perfectly competitive market, it finds a solution satisfying market clearance, where every resource has been allocated. Unfortunately, this exact property makes it possible to cheat CEEI: a user can claim she needs more of some underutilized resource even when she does not, leading CEEI to give more tasks overall to this user to achieve market clearance.

6 Analysis

In this section, we discuss which of the properties presented in Section 3 are satisfied by Asset Fairness, CEEI, and DRF. We also evaluate the accuracy of DRF when task sizes do not match the available resources exactly.

6.1 Fairness Properties

Table 2 summarizes the fairness properties that are satisfied by Asset Fairness, CEEI, and DRF. The Appendix contains the proofs of the main properties of DRF, while our technical report [14] contains a more complete list of results for DRF and CEEI. In the remainder of this section, we discuss some of the interesting missing entries in the table, *i.e.*, properties violated by each of these disciplines. In particular, we show through examples why Asset Fairness and CEEI lack the properties that they

Property	Allocation Policy		
	Asset	CEEI	DRF
Sharing Incentive		✓	✓
Strategy-proofness	✓		✓
Envy-freeness	✓	✓	✓
Pareto efficiency	✓	✓	✓
Single Resource Fairness	✓	✓	✓
Bottleneck Fairness		✓	✓
Population Monotonicity	✓		✓
Resource Monotonicity			

Table 2: Properties of Asset Fairness, CEEI and DRF.

do, and we prove that no policy can provide resource monotonicity without violating either sharing incentive or Pareto efficiency to explain why DRF lacks resource monotonicity.

6.1.1 Properties Violated by Asset Fairness

While being the simplest policy, Asset Fairness violates several important properties: sharing incentive, bottleneck fairness, and resource monotonicity. Next, we use examples to show the violation of these properties.

Theorem 1 *Asset Fairness violates the sharing incentive property.*

Proof Consider the following example, illustrated in Figure 5: two users in a system with $\langle 30, 30 \rangle$ total resources have demand vectors $D_1 = \langle 1, 3 \rangle$, and $D_2 = \langle 1, 1 \rangle$. Asset fairness will allocate the first user 6 tasks and the second user 12 tasks. The first user will receive $\langle 6, 18 \rangle$ resources, while the second will use $\langle 12, 12 \rangle$. While each user gets an equal aggregate share of $\frac{24}{60}$, the second user gets less than half (15) of both resources. This violates the sharing incentive property, as the second user would be better off to statically partition the cluster and own half of the nodes. \square

Theorem 2 *Asset Fairness violates the bottleneck fairness property.*

Proof Consider a scenario with a total resource vector of $\langle 21, 21 \rangle$ and two users with demand vectors $D_1 = \langle 3, 2 \rangle$ and $D_2 = \langle 4, 1 \rangle$, making resource 1 the bottleneck resource. Asset fairness will give each user 3 tasks, equalizing their aggregate usage to 15. However, this only gives the first user $\frac{3}{7}$ of resource 1 (the contended bottleneck resource), violating bottleneck fairness. \square

Theorem 3 *Asset fairness does not satisfy resource monotonicity.*

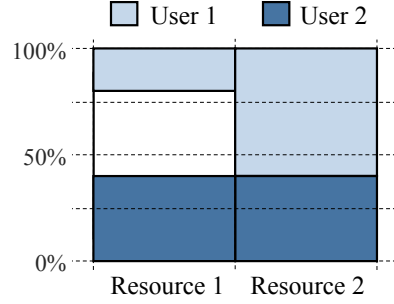


Figure 5: Example showing that Asset Fairness can fail to meet the sharing incentive property. Asset Fairness gives user 2 less than half of both resources.

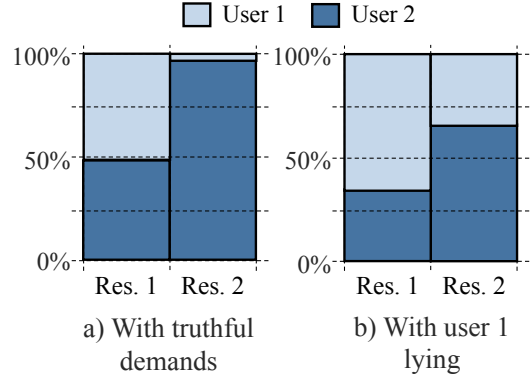


Figure 6: Example showing how CEEI violates strategy proofness. User 1 can increase her share by claiming that she needs more of resource 2 than she actually does.

Proof Consider two users A and B with demands $\langle 4, 2 \rangle$ and $\langle 1, 1 \rangle$ and 77 units of two resources. Asset fairness allocates A a total of $\langle 44, 22 \rangle$ and B $\langle 33, 33 \rangle$ equalizing their sum of shares to $\frac{66}{77}$. If resource two is doubled, both users' share of the second resource is halved, while the first resource is saturated. Asset fairness now decreases A 's allocation to $\langle 42, 21 \rangle$ and increases B 's to $\langle 35, 35 \rangle$, equalizing their shares to $\frac{42}{77} + \frac{21}{154} = \frac{35}{77} + \frac{35}{154} = \frac{105}{154}$. Thus resource monotonicity is violated. \square

6.1.2 Properties Violated by CEEI

While CEEI is envy-free and Pareto efficient, it turns out that it is *not* strategy proof. Intuitively, this is because CEEI assumes a perfectly competitive market that achieves *market clearance*, i.e., matching of supply and demand and allocation of all the available resources. This can lead to CEEI giving much higher shares to users that use more of a less-contended resource in order to fully utilize that resource. Thus, a user can claim that she needs more of some underutilized resource to increase her overall share of resources. We illustrate this below.

Theorem 4 *CEEI is not strategy-proof.*

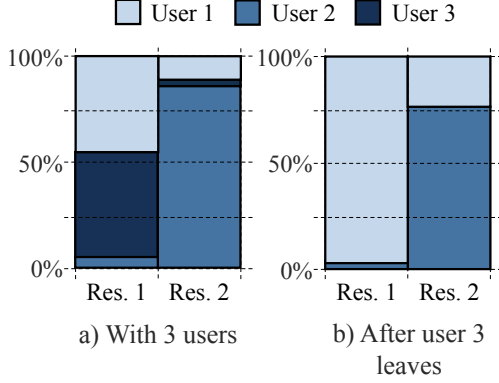


Figure 7: Example showing that CEEI violates population monotonicity. When user 3 leaves, CEEI changes the allocation from a) to b), lowering the share of user 2.

Proof Consider the following example, shown in Figure 6. Assume a total resource vector of $\langle 100, 100 \rangle$, and two users with demands $\langle 16, 1 \rangle$ and $\langle 1, 2 \rangle$. In this case, CEEI allocates $\frac{100}{31}$ and $\frac{1500}{31}$ tasks to each user respectively (approximately 3.2 and 48.8 tasks). If user 1 changes her demand vector to $\langle 16, 8 \rangle$, asking for more of resource 2 than she actually needs, CEEI gives the users $\frac{25}{6}$ and $\frac{100}{3}$ tasks respectively (approximately 4.2 and 33.3 tasks). Thus, user 1 improves her number of tasks from 3.2 to 4.2 by lying about her demand vector. User 2 suffers because of this, as her task allocation decreases. \square

In addition, for the same intuitive reason (market clearance), we have the following result:

Theorem 5 *CEEI violates population monotonicity.*

Proof Consider the total resource vector $\langle 100, 100 \rangle$ and three users with the following demand vectors $D_1 = \langle 4, 1 \rangle$, $D_2 = \langle 1, 16 \rangle$, and $D_3 = \langle 16, 1 \rangle$ (see Figure 7). CEEI will yield the allocation $A_1 = \langle 11.3, 5.4, 3.1 \rangle$, where the numbers in parenthesis represent the number of tasks allocated to each user. If user 3 leaves the system and relinquishes her resource, CEEI gives the new allocation $A_2 = \langle 23.8, 4.8 \rangle$, which has made user 2 worse off than in A_1 . \square

6.1.3 Resource Monotonicity vs. Sharing Incentives and Pareto efficiency

As shown in Table 2, DRF achieves all the properties except resource monotonicity. Rather than being a limitation of DRF, this is a consequence of the fact that sharing incentive, Pareto efficiency, and resource monotonicity cannot be achieved simultaneously. Since we consider the first two of these properties to be more important (see Section 3) and since adding new resources to a system is a relatively rare event, we chose to satisfy sharing incentive and Pareto efficiency, and give up resource monotonicity. In particular, we have the following result.

Theorem 6 *No allocation policy that satisfies the sharing incentive and Pareto efficiency properties can also satisfy resource monotonicity.*

Proof We use a simple example to prove this property. Consider two users A and B with symmetric demands $\langle 2, 1 \rangle$, and $\langle 1, 2 \rangle$, respectively, and assume equal amounts of both resources. Sharing incentive requires that user A gets at least half of resource 1 and user B gets half of resource 2. By Pareto efficiency, we know that at least one of the two users must be allocated more resources. Without loss of generality, assume that user A is given more than half of resource 1 (a symmetric argument holds if user B is given more than half of resource 2). If the total amount of resource 2 is now increased by a factor of 4, user B is no longer getting its guaranteed share of half of resource 2. Now, the only feasible allocation that satisfies the sharing incentive is to give both users half of resource 1, which would require decreasing user 1's share of resource 1, thus violating resource monotonicity. \square

This theorem explains why both DRF and CEEI violate resource monotonicity.

6.2 Discrete Resource Allocation

So far, we have implicitly assumed one big resource pool whose resources can be allocated in arbitrarily small amounts. Of course, this is often not the case in practice. For example, clusters consist of many small machines, where resources are allocated to tasks in discrete amounts. In the remainder of this section, we refer to these two scenarios as the *continuous*, and the *discrete* scenario, respectively. We now turn our attention to how fairness is affected in the discrete scenario.

Assume a cluster consisting of K machines. Let max-task denote the maximum demand vector across all demand vectors, i.e., $\text{max-task} = \langle \max_i \{d_{i,1}\}, \max_i \{d_{i,2}\}, \dots, \max_i \{d_{i,m}\} \rangle$. Assume further that any task can be scheduled on every machine, i.e., the total amount of resources on each machine is at least max-task . We only consider the case when each user has strictly positive demands. Given these assumptions, we have the following result.

Theorem 7 *In the discrete scenario, it is possible to allocate resources such that the difference between the allocations of any two users is bounded by one max-task compared to the continuous allocation scenario.*

Proof Assume we start allocating resources on one machine at a time, and that we always allocate a task to the user with the lowest dominant share. As long as there is at least a max-task available on the first machine, we

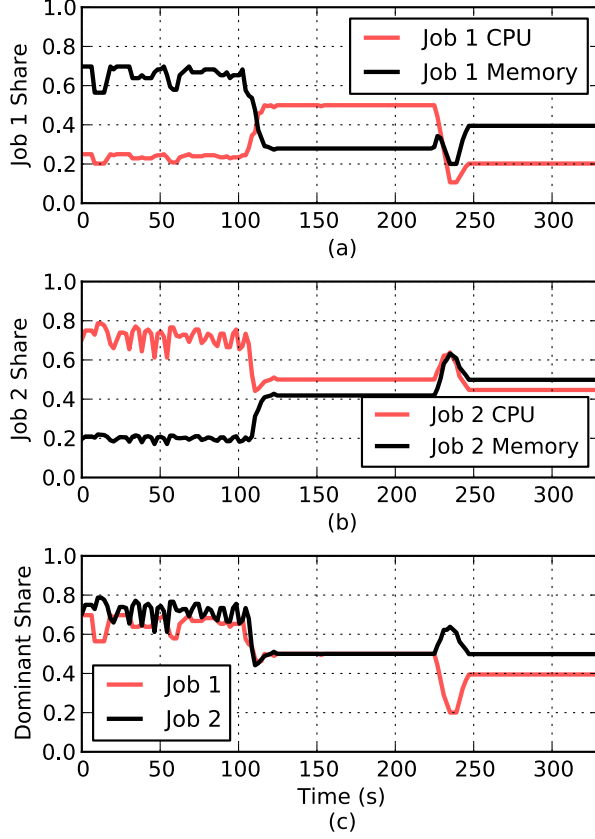


Figure 8: CPU, memory and dominant share for two jobs.

continue to allocate a task to the next user with least dominant share. Once the available resources on the first machine become less than a *max-task* size, we move to the next machine and repeat the process. When the allocation completes, the difference between two user’s allocations of their dominant resources compared to the continuous scenario is at most *max-task*. If this were not the case, then some user *A* would have more than *max-task* discrepancy w.r.t. to another user *B*. However, this cannot be the case, because the last time *A* was allocated a task, *B* should have been allocated a task instead. \square

7 Experimental Results

This section evaluates DRF through micro- and macro-benchmarks. The former is done through experiments running an implementation of DRF in the Mesos cluster resource manager [16]. The latter is done using trace-driven simulations.

We start by showing how DRF dynamically adjusts the shares of jobs with different resource demands in Section 7.1. In Section 7.2, we compare DRF against slot-level fair sharing (as implemented by Hadoop Fair Scheduler [34] and Quincy [18]), and CPU-only fair sharing. Finally, in Section 7.3, we use Facebook traces to compare DRF and the Hadoop’s Fair Scheduler in terms of utiliza-

tion and job completion time.

7.1 Dynamic Resource Sharing

In our first experiment, we show how DRF dynamically shares resources between jobs with different demands. We ran two jobs on a 48-node Mesos cluster on Amazon EC2, using “extra large” instances with 4 CPU cores and 15 GB of RAM. We configured Mesos to allocate up to 4 CPUs and 14 GB of RAM on each node, leaving 1 GB for the OS. We submitted two jobs that launched tasks with different resource demands at different times during a 6-minute interval.

Figures 8 (a) and 8 (b) show the CPU and memory allocations given to each job as a function of time, while Figure 8 (c) shows their dominant shares. In the first 2 minutes, job 1 uses $\langle 1 \text{ CPU}, 10 \text{ GB RAM} \rangle$ per task and job 2 uses $\langle 1 \text{ CPU}, 1 \text{ GB RAM} \rangle$ per task. Job 1’s dominant resource is RAM, while job 2’s dominant resource is CPU. Note that DRF equalizes the jobs’ shares of their dominant resources. In addition, because jobs have different dominant resources, their dominant shares exceed 50%, i.e., job 1 uses around 70% of the RAM while job 2 uses around 75% of the CPUs. Thus, the jobs benefit from running in a shared cluster as opposed to taking half the nodes each. This captures the essence of the sharing incentive property.

After 2 minutes, the task sizes of both jobs change, to $\langle 2 \text{ CPUs}, 4 \text{ GB} \rangle$ for job 1 and $\langle 1 \text{ CPU}, 3 \text{ GB} \rangle$ for job 2. Now, both jobs’ dominant resource is CPU, so DRF equalizes their CPU shares. Note that DRF switches allocations dynamically by having Mesos offer resources to the job with the smallest dominant share as tasks finish.

Finally, after 2 more minutes, the task sizes of both jobs change again: $\langle 1 \text{ CPU}, 7 \text{ GB} \rangle$ for job 1 and $\langle 1 \text{ CPU}, 4 \text{ GB} \rangle$ for job 2. Both jobs’ dominant resource is now memory, so DRF tries to equalize their memory shares. The reason the shares are not exactly equal is due to resource fragmentation (see Section 6.2).

7.2 DRF vs. Alternative Allocation Policies

We next evaluate DRF with respect to two alternative schemes: slot-based fair scheduling (a common policy in current systems, such as the Hadoop Fair Scheduler [34] and Quincy [18]) and (max-min) fair sharing applied only to a single resource (CPU). For the experiment, we ran a 48-node Mesos cluster on EC2 instances with 8 CPU cores and 7 GB RAM each. We configured Mesos to allocate 8 CPUs and 6 GB RAM on each node, leaving 1 GB free for the OS. We implemented these three scheduling policies as Mesos allocation modules.

We ran a workload with two classes of users, representing two organizational entities with different workloads. One of the entities had four users submitting small jobs with task demands $\langle 1 \text{ CPU}, 0.5 \text{ GB} \rangle$. The other en-

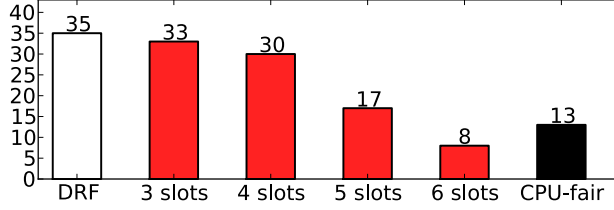


Figure 9: Number of large jobs completed for each allocation scheme in our comparison of DRF against slot-based fair sharing and CPU-only fair sharing.

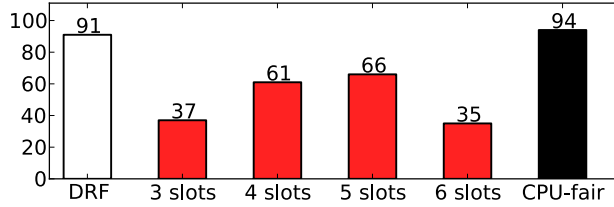


Figure 10: Number of small jobs completed for each allocation scheme in our comparison of DRF against slot-based fair sharing and CPU-only fair sharing.

tity had four users submitting large jobs with task demands (2 CPUs, 2 GB). Each job consisted of 80 tasks. As soon as a job finished, the user would launch another job with similar demands. Each experiment ran for ten minutes. At the end, we computed the number of completed jobs of each type, as well as their response times.

For the slot-based allocation scheme, we varied the number of slots per machine from 3 to 6 to see how it affected performance. Figures 9 through 12 show our results. In Figures 9 and 10, we compare the number of jobs of each type completed for each scheduling scheme in ten minutes. In Figures 11 and 12, we compare average response times.

Several trends are apparent from the data. First, with slot-based scheduling, both the throughput and job response times are worse than with DRF, regardless of the number of slots. This is because with a low slot count, the scheduler can undersubscribe nodes (*e.g.*, launch only 3 small tasks on a node), while with a large slot count, it can oversubscribe them (*e.g.*, launch 4 large tasks on a node and cause swapping because each task needs 2 GB and the node only has 6 GB). Second, with fair sharing at the level of CPUs, the number of small jobs executed is similar to DRF, but there are much fewer large jobs executed, because memory is overcommitted on some machines and leads to poor performance for all the high-memory tasks running there. Overall, the DRF-based scheduler that is aware of both resources has the lowest response times and highest overall throughput.

7.3 Simulations using Facebook Traces

Next we use log traces from a 2000-node cluster at Facebook, containing data for a one week period (October

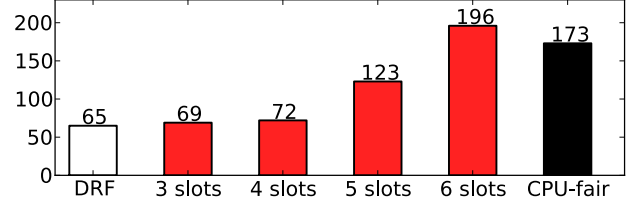


Figure 11: Average response time (in seconds) of large jobs for each allocation scheme in our comparison of DRF against slot-based fair sharing and CPU-only fair sharing.

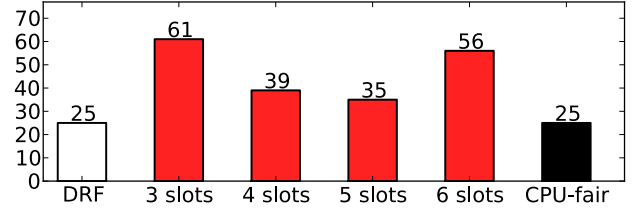


Figure 12: Average response time (in seconds) of small jobs for each allocation scheme in our comparison of DRF against slot-based fair sharing and CPU-only fair sharing.

2010). The data consists of Hadoop MapReduce jobs. We assume task duration, CPU usage, and memory consumption is identical as in the original trace. The traces are simulated on a smaller cluster of 400 nodes to reach higher utilization levels, such that fairness becomes relevant. Each node in the cluster consists of 12 slots, 16 cores, and 32 GB memory. Figure 13 shows a short 300 second sub-sample to visualize how CPU and memory utilization looks for the same workload when using DRF compared to Hadoop’s fair scheduler (slot). As shown in the figure, DRF provides higher utilization, as it is able to better match resource allocations with task demands.

Figure 14 shows the reduction of the average job completion times for DRF as compared to the Hadoop fair scheduler. The workload is quite heavy on small jobs, which experience no improvements (*i.e.*, -3%). This is because small jobs typically consist of a single execution phase, and the completion time is dominated by the longest task. Thus completion time is hard to improve for such small jobs. In contrast, the completion times of the larger jobs reduce by as much as 66%. This is because these jobs consists of many phases, and thus they can benefit from the higher utilization achieved by DRF.

8 Related Work

We briefly review related work in computer science and economics.

While many papers in computer science focus on multi-resource fairness, they are only considering multiple instances of the *same* interchangeable resource, *e.g.*, CPU [6, 7, 35], and bandwidth [10, 20, 21]. Unlike these approaches, we focus on the allocation of resources of different types.

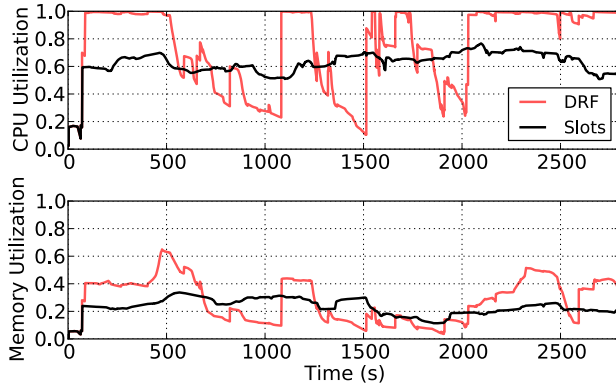


Figure 13: CPU and memory utilization for DRF and slot fairness for a trace from a Facebook Hadoop cluster.

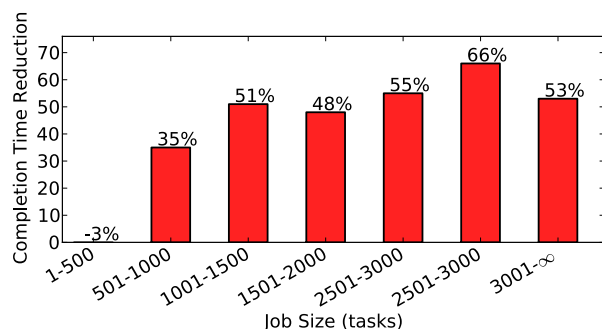


Figure 14: Average reduction of the completion times for different job sizes for a trace from a Facebook Hadoop cluster.

Quincy [18] is a scheduler developed in the context of the Dryad cluster computing framework [17]. Quincy achieves fairness by modeling the fair scheduling problem as a min-cost flow problem. Quincy does not currently support multi-resource fairness. In fact, as mentioned in the discussion section of the paper [18, pg. 17], it appears difficult to incorporate multi-resource requirements into the min-cost flow formulation.

Hadoop currently provides two fair sharing schedulers [1, 2, 34]. Both these schedulers allocate resources at the slot granularity, where a slot is a fixed fraction of the resources on a machine. As a result, these schedulers cannot always match the resource allocations with the tasks' demands, especially when these demands are widely heterogeneous. As we have shown in Section 7, this mismatch may lead to either low cluster utilization or poor performance due to resource oversubscription.

In the microeconomic literature, the problem of equity has been studied within and outside of the framework of game theory. The books by Young [33] and Moulin [22] are entirely dedicated to these topics and provide good introductions. The preferred method of fair division in microeconomics is CEEI [3, 33, 22], as introduced by Varian [30]. We have therefore devoted considerable attention to it in Section 5.2. CEEI's main drawback com-

pared to DRF is that it is not strategy-proof. As a result, users can manipulate the scheduler by lying about their demands.

Many of the fair division policies proposed in the microeconomics literature are based on the notion of utility and, hence, focus on the single metric of utility. In the economics literature, max-min fairness is known as the lexicographic ordering [26, 25] (leximin) of utilities.

The question is what the user utilities are in the multi-resource setting, and how to compare such utilities. One natural way is to define utility as the number of tasks allocated to a user. But modeling utilities this way, together with leximin, violates many of the fairness properties we proposed. Viewed in this light, DRF makes two contributions. First, it suggests using the dominant share as a proxy for utility, which is equalized using the standard leximin ordering. Second, we prove that this scheme is strategy-proof for such utility functions. Note that the leximin ordering is a lexicographic version of the Kalai-Smorodinsky (KS) solution [19]. Thus, our result shows that KS is strategy-proof for such utilities.

9 Conclusion and Future Work

We have introduced Dominant Resource Fairness (DRF), a fair sharing model that generalizes max-min fairness to multiple resource types. DRF allows cluster schedulers to take into account the heterogeneous demands of data-center applications, leading to both fairer allocation of resources and higher utilization than existing solutions that allocate identical resource slices (slots) to all tasks. DRF satisfies a number of desirable properties. In particular, DRF is strategy-proof, so that users are incentivized to report their demands accurately. DRF also incentivizes users to share resources by ensuring that users perform at least as well in a shared cluster as they would in smaller, separate clusters. Other schedulers that we investigated, as well as alternative notions of fairness from the microeconomic literature, fail to satisfy all of these properties.

We have evaluated DRF by implementing it in the Mesos resource manager, and shown that it can lead to better overall performance than the slot-based fair schedulers that are commonly in use today.

9.1 Future Work

There are several interesting directions for future research. First, in cluster environments with discrete tasks, one interesting problem is to minimize resource fragmentation without compromising fairness. This problem is similar to bin-packing, but where one must pack as many items (tasks) as possible subject to meeting DRF. A second direction involves defining fairness when tasks have placement constraints, such as machine preferences. Given the current trend of multi-core machines,

a third interesting research direction is to explore the use of DRF as an operating system scheduler. Finally, from a microeconomic perspective, a natural direction is to investigate whether DRF is the only possible strategy-proof policy for multi-resource fairness, given other desirable properties such as Pareto efficiency.

10 Acknowledgements

We thank Eric J. Friedman, Hervé Moulin, John Wilkes, and the anonymous reviewers for their invaluable feedback. We thank Facebook for making available their traces. This research was supported by California MICRO, California Discovery, the Swedish Research Council, the Natural Sciences and Engineering Research Council of Canada, a National Science Foundation Graduate Research Fellowship,⁵ and the RAD Lab sponsors: Google, Microsoft, Oracle, Amazon, Cisco, Cloudera, eBay, Facebook, Fujitsu, HP, Intel, NetApp, SAP, VMware, and Yahoo!.

References

- [1] Hadoop Capacity Scheduler.
http://hadoop.apache.org/common/docs/r0.20.2/capacity_scheduler.html.
- [2] Hadoop Fair Scheduler.
http://hadoop.apache.org/common/docs/r0.20.2/fair_scheduler.html.
- [3] Personal communication with Hervé Moulin.
- [4] A. K. Agrawala and R. M. Bryant. Models of memory scheduling. In *SOSP '75*, 1975.
- [5] J. Axboe. Linux Block IO – Present and Future (Completely Fair Queueing). In *Ottawa Linux Symposium 2004*, pages 51–61, 2004.
- [6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.
- [7] S. K. Baruah, J. Gehrke, and C. G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *IPPS '95*, 1995.
- [8] J. Bennett and H. Zhang. WF²Q: Worst-case fair weighted fair queueing. In *INFOCOM*, 1996.
- [9] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, second edition, 1992.
- [10] J. M. Blanquer and B. Özden. Fair queueing for aggregated multiple links. *SIGCOMM '01*, 31(4):189–197, 2001.
- [11] B. Caprita, W. C. Chan, J. Nieh, C. Stein, and H. Zheng. Group ratio round-robin: O(1) proportional share scheduling for uniprocessor and multiprocessor systems. In *USENIX Annual Technical Conference*, 2005.
- [12] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM '89*, pages 1–12, New York, NY, USA, 1989. ACM.
- [13] D. Foley. Resource allocation and the public sector. *Yale Economic Essays*, 7(1):73–76, 1967.
- [14] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. Technical Report UCB/EECS-2011-18, EECS Department, University of California, Berkeley, Mar 2011.
- [15] P. Goyal, H. Vin, and H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Transactions on Networking*, 5(5):690–704, Oct. 1997.
- [16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.
- [17] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys 07*, 2007.
- [18] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *SOSP '09*, 2009.
- [19] E. Kalai and M. Smorodinsky. Other Solutions to Nash's Bargaining Problem. *Econometrica*, 43(3):513–518, 1975.
- [20] J. M. Kleinberg, Y. Rabani, and É. Tardos. Fairness in routing and load balancing. *J. Comput. Syst. Sci.*, 63(1):2–20, 2001.
- [21] Y. Liu and E. W. Knightly. Opportunistic fair scheduling over multiple wireless channels. In *INFOCOM*, 2003.
- [22] H. Moulin. *Fair Division and Collective Welfare*. The MIT Press, 2004.
- [23] J. Nash. The Bargaining Problem. *Econometrica*, 18(2):155–162, April 1950.
- [24] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. *ACM/IEEE Transactions on Networking*, 1(3):344–357, June 1993.
- [25] E. A. Pazner and D. Schmeidler. Egalitarian equivalent allocations: A new concept of economic equity. *Quarterly Journal of Economics*, 92:671–687, 1978.
- [26] A. Sen. Rawls Versus Bentham: An Axiomatic Examination of the Pure Distribution Problem. *Theory and Decision*, 4(1):301–309, 1974.
- [27] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. *IEEE Trans. Net*, 1996.
- [28] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *IEEE RTSS 96*, 1996.
- [29] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *SIGCOMM*, 1998.
- [30] H. Varian. Equity, envy, and efficiency. *Journal of Economic Theory*, 9(1):63–91, 1974.
- [31] C. A. Waldspurger. *Lottery and Stride Scheduling: Flexible Proportional Share Resource Management*. PhD thesis, MIT, Laboratory of Computer Science, Sept. 1995. MIT/LCS/TR-667.
- [32] C. A. Waldspurger and W. E. Weihl. Lottery scheduling:

⁵Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

flexible proportional-share resource management. In *OSDI '94*, 1994.

- [33] H. P. Young. *Equity: in theory and practice*. Princeton University Press, 1994.
- [34] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *EuroSys 10*, 2010.
- [35] D. Zhu, D. Mossé, and R. G. Melhem. Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary? In *IEEE RTSS*, 2003.

A Appendix: DRF Properties

In this appendix, we present the main properties of DRF. The technical report [14] contains a more complete list of results for DRF and CEEI. For context, the following table summarizes the properties satisfied by Asset Fairness, CEEI, and DRF, respectively.

In this section, we assume that all users have an unbounded number of tasks. In addition, we assume that all tasks of a user have the same demand vector, and we will refer to this vector as the user's demand vector.

Next, we present *progressive filling* [9], a simple technique to achieve DRF allocation when all resources are arbitrary divisible. This technique is instrumental in proving our results.

A.1 Progressive Filling for DRF

Progressive filling is an idealized algorithm to achieve max-min fairness in a system in which resources can be allocated in arbitrary small amounts [9, pg 450]. It was originally used in a networking context, but we now adapt it to our problem domain. In the case of DRF, progressive filling increases all users' dominant shares at the same rate, while increasing their other resource allocations *proportionally* to their task demand vectors, until at least one resource is saturated. At this point, the allocations of all users using the saturated resource are frozen, and progressive filling continues recursively after eliminating these users. In this case, progressive filling terminates when there are no longer users whose dominant shares can be increased.

Progressive filling for DRF is equivalent to the scheduling algorithm presented in Figure 1 after appropriately scaling the users' demand vectors. In particular, each user's demand vector is scaled such that allocating resources to a user according to her scaled demand vector will increase her dominant share by a fixed ϵ , which is the same for all users. Let $D_i = \langle d_{i,1}, d_{i,2}, \dots, d_{i,m} \rangle$ be the demand vector of user i , let r_k be her dominant share⁶, and let $s_i = \frac{d_{i,k}}{r_k}$ be her dominant share. We then scale the demand vector of user i by $\frac{\epsilon}{s_i}$, i.e., $D'_i = \frac{\epsilon}{s_i} D_i = \frac{\epsilon}{s_i} \langle d_{i,1}, d_{i,2}, \dots, d_{i,m} \rangle$. Thus, every time

⁶Recall that in this section we assume that all tasks of a user have the same demand vector.

a task of user i is selected, she is allocated an amount $\frac{\epsilon}{s_i} d_{i,k} = \epsilon \cdot r_k$ of the dominant resource. This means that the share of the dominant resource of user i increases by $(\epsilon \cdot r_k)/r_k = \epsilon$, as expected.

A.2 Allocation Properties

We start with a preliminary result.

Lemma 8 *Every user in a DRF allocation has at least one saturated resource.*

Proof Assume this is not the case, i.e., none of the resources used by user i is saturated. However, this contradicts the assumption that progressive filling has completed the computation of the DRF allocation. Indeed, as long as none of the resources of user i are saturated, progressive filling will continue to increase the allocations of user i (and of all the other users sharing only non-saturated resources). \square

Recall that progressive filling always allocates the resources to a user *proportionally* to the user's demand vector. More precisely, let $D_i = \langle d_{i,1}, d_{i,2}, \dots, d_{i,m} \rangle$ be the demand vector of user i . Then, at any time t during the progressive filling process, the allocation of user i is proportional to the demand vector,

$$A_i(t) = \alpha_i(t) \cdot D_i = \alpha_i(t) \cdot \langle d_{i,1}, d_{i,2}, \dots, d_{i,m} \rangle \quad (1)$$

where $\alpha_i(t)$ is a positive scalar.

Now, we are in position to prove the DRF properties.

Theorem 9 *DRF is Pareto efficient.*

Proof Assume user i can increase her dominant share, s_i , without decreasing the dominant share of anyone else. According to Lemma 8, user i has at least one saturated resource. If no other user is using the saturated resource, then we are done as it would be impossible to increase i 's share of the saturated resource. If other users are using the saturated resource, then increasing the allocation of i would result in decreasing the allocation of at least another user j sharing the same saturated resource. Since under progressive filling, the resources allocated by any user are proportional to her demand vector (see Eq. 1), decreasing the allocation of any resource used by user i will also decrease i 's dominant share. This contradicts our hypothesis, and therefore proves the result. \square

Theorem 10 *DRF satisfies the sharing incentive and bottleneck fairness properties.*

Proof Consider a system consisting of n users. Assume resource k is the first one being saturated by using progressive filling. Let i be the user allocating the largest share on resource k , and let $t_{i,k}$ denote her share of k . Since resource k is saturated, we have trivially $t_{i,k} \geq \frac{1}{n}$.

Furthermore, by the definition of the dominant share, we have $s_i \geq t_{i,k} \geq \frac{1}{n}$. Since progressive filling increases the allocation of each user's dominant resource at the *same* rate, it follows that each user gets at least $\frac{1}{n}$ of her dominant resource. Thus, DRF satisfies the sharing incentive property. If all users have the same dominant resource, each user gets exactly $\frac{1}{n}$ of that resource. As a result, DRF satisfies the bottleneck fairness property as well. \square

Theorem 11 *Every DRF allocation is envy-free.*

Proof Assume by contradiction that user i envies another user j . For user i to envy another user j , user j must have a strictly higher share of every resource that i wants; otherwise i cannot run more tasks under j 's allocation. This means that user j 's dominant share is strictly larger than user i 's dominant share. Since every resource allocated to user i is also allocated to user j , this means that user j cannot reach its saturated resource *after* user i , i.e., $t_j \leq t_i$, where t_k is the time that user k 's allocation gets frozen due to saturation. However, if $t_j \leq t_i$, under progressive filling, the dominant shares of users j and i will be equal at time t_j , after which the dominant share of user i can only increase, violating the hypothesis. \square

Theorem 12 (Strategy-proofness) *A user cannot increase her dominant share in DRF by altering her true demand vector.*

Proof Assume user i can increase her dominant share by using a demand vector $\hat{d}_i \neq d_i$. Let $a_{i,j}$ and $\hat{a}_{i,j}$ denote the amount of resource j user i is allocated using progressive filling when the user uses the vector d_i and \hat{d}_i , respectively. For user i to be better off using \hat{d}_i , we need that $\hat{a}_{i,k} > a_{i,k}$ for every resource k where $d_{i,k} > 0$. Let r denote the first resource that becomes saturated for user i when she uses the demand vector d_i . If no other user is allocated resource r ($a_{j,r} = 0$ for all $j \neq i$), this contradicts the hypothesis as user i is already allocated the entire resource r , and thus cannot increase her allocation of r using another demand vector \hat{d}_i . Thus, assume there are other users that have been allocated r ($a_{j,r} > 0$ for some $j \neq i$). In this case, progressive filling will eventually saturate r at time t when using d_i , and at time t' when using demand \hat{d}_i . Recall that the dominant share is the maximum of a user's shares, thus i must have a higher dominant share in the allocation \hat{a} than in a . Thus, $t' > t$, as progressive filling increases the dominant share at a constant rate. This implies that i —when using \hat{d} —does not saturate any resource before time t' , and hence does not affect other user's allocation before time t' . Thus, when i uses \hat{d} , any user m using resource r has allocation $a_{m,r}$ at time t . Therefore, at time t , there is only $a_{i,r}$ amount of r left for user i , which contradicts the assumption that $\hat{a}_{i,r} > a_{i,r}$. \square

The strategy-proofness of DRF shows that a user will not be better off by demanding resources that she does not need. The following example shows that excess demand can in fact hurt user's allocation, leading to a lower dominant share. Consider a cluster with two resources, and 10 users, the first with demand vector $\langle 1, 0 \rangle$ and the rest with demand vectors $\langle 0, 1 \rangle$. The first user gets the entire first resource, while the rest of the users each get $\frac{1}{9}$ of the second resource. If user 1 instead changes her demand vector to $\langle 1, 1 \rangle$, she can only be allocated $\frac{1}{10}$ of each resource and the rest of the users get $\frac{1}{10}$ of the second resource.

In practice, the situation can be exacerbated as resources in datacenters are typically partitioned across different physical machines, leading to fragmentation. Increasing one's demand artificially might lead to a situation in which, while there are enough resources on the whole, there are not enough on any single machine to satisfy the new demand. See Section 6.2 for more information.

Next, for simplicity we assume *strictly positive demand vectors*, i.e., the demand of every user for every resource is non-zero.

Theorem 13 *Given strictly positive demand vectors, DRF guarantees that every user gets the same dominant share, i.e., every DRF allocation ensures $s_i = s_j$, for all users i and j .*

Proof Progressive filling will start increasing every users' dominant resource allocation at the same rate until one of the resources becomes saturated. At this point, no more resources can be allocated to any user as every user demands a positive amount of the saturated resource. \square

Theorem 14 *Given strictly positive demands, DRF satisfies population monotonicity.*

Proof Consider any DRF allocation. Non-zero demands imply that all users have the same saturated resource(s). Consider removing a user and relinquishing her currently allocated resources, which is some amount of every resource. Since all users have the same dominant share α , any new allocation which decreases any user i 's dominant share below α would, due to Pareto efficiency, have to allocate another user j a dominant share of more than α . The resulting allocation would violate max-min fairness, as it would be possible to increase i 's dominant share by decreasing the allocation of j , who already has a higher dominant share than i . \square

However, we note that in the absence of strictly positive demand vectors, DRF no longer satisfies the population monotonicity property [14].