Dominant Resource Fairness: Fair Allocation of Heterogeneous Resources in Datacenters

Ali Ghodsi
Matei Zaharia
Benjamin Hindman
Andrew Konwinski
Scott Shenker
Ion Stoica

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-55
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-55.html

May 7, 2010

Acknowledgement

This research was supported by California MICRO, California Discovery, the Natural Sciences and Engineering Research Council of Canada, as well as the following Berkeley RAD Lab sponsors: Sun Microsystems, Google, Microsoft, Amazon, Cisco, Cloudera, eBay, Facebook, Fujitsu, HP, Intel, NetApp, SAP, VMware, and Yahoo!.

Dominant Resource Fairness: Fair Allocation of Heterogeneous Resources in Datacenters

Ali Ghodsi
alig@cs.berkeley.edu
U.C. Berkeley

Matei Zaharia
matei@cs.berkeley.edu
U.C. Berkeley

Benjamin Hindman
benh@cs.berkeley.edu
U.C. Berkeley

Andy Konwinski
andyk@cs.berkeley.edu
U.C. Berkeley

Scott Shenker
shenker@icsi.berkeley.edu
U.C. Berkeley

Ion Stoica
istoica@cs.berkeley.edu
U.C. Berkeley

May 7, 2010

Abstract

This paper investigates how different resources can be *fairly* allocated among users that possibly prioritize them differently. We introduce a fairness policy, called Dominant Resource Fairness (DRF), which is an adaptation of max-min fairness from networking to datacenter environments. We show that DRF, unlike other policies which we investigated, satisfies a number of desirable properties that a fair datacenter scheduler should have, including guaranteeing that every user gets $\frac{1}{n}$ of some resource and that users can relinquish resources without hurting other users' allocations. DRF is also envy-free, incentivizing users to correctly report their resource demand. When compared to other intuitive schedulers, as well as competing ones from microeconomic theory, DRF is more fair.

1 Introduction

Cloud computing enables the execution and multiplexing of many heterogeneous tasks in a common datacenter. A centerpiece of any cloud framework is the scheduler, which makes decisions about which task to execute on which compute node. In doing so, it is desirable for the scheduler to provide some fairness guarantees when tasks compete for the same resources.

In many datacenters, different types of tasks compete for different resources. For example, some tasks are CPU-bound, while others are disk or bandwidth bound. Nevertheless, most tasks, regardless of their bounding resource, need to use *some* amount of other resources too. In other words, task owners prioritize resources differently. This paper investigates how fairness can be achieved in such environments. More precisely, we investigate how fairness can be achieved when scheduling multiple tasks, each with possibly different demands across a set of different resources.

This paper proposes a notion of fairness, called Dominant Resource Fairness (DRF) for datacenter environments and provides an efficient algorithm that implements it. We extensively compare DRF with other fairness policies as well as related work in microeconomic theory. We show that other fairness policies are less desirable in the datacenter context. The proposed fairness property is intuitive, and can be seen as an adaptation of Max-min fairness, while Jain's Fairness Index (JFI) can be used to quantify its level of fairness. DRF is also strategy-proof, incentivizing users to not misreport their resource demands.

Outline. Section 2 introduces our proposed fairness policy, and shows its relation to Max/Min fairness, as well as Jain's Fairness Index. We list desirable fairness properties in Section 2.5, and use them in Section 3.3 to compare DRF with other fairness policies as well as with competing fair division methods in microeconomic literature. Section 4 concludes.

2 Dominant Resource Fairness

We begin by intuitively describing Dominant Resource Fairness (DRF) using a simple example. Then we give a more precise model and formally describe the advantages of DRF.

The intuition behind DRF is that users care about the number of tasks they are allocated. In other words, users care about the resource that they relatively demand most of, since that is the resource they will be allocated most of. We define a user's *dominant resource* to be the resource that it *percentage-wise* demands most of, *e.g.,* with a total 10 CPUs and 40GB memory, a user that demands 1 CPU and 2 GB memory per task has CPU as its dominant resource, as that is the resource that dominates its relative demand. Naturally, a node might have multiple dominant resources.

Consider a cluster with 12 CPUs and 12 GB memory, and two users. The first user wants to run tasks that use 4 CPUs and 1 GB memory, while the second user wants to run tasks that use 1 CPU and 2 GB memory. We assume, for this example, that each user has infinite task demand. *Dominant Resource Fairness* (DRF) attempts to give all users an equal amount of their dominant resources. In the given example, DRF will allocate the two users 2 and 4 tasks, respectively (see Figure 1). Thus, the resource usage of the two users becomes, $\langle 8, 2\rangle$, and $\langle 4, 8\rangle$, respectively. Hence, both users get $\frac{2}{3}$ of their dominant resource.

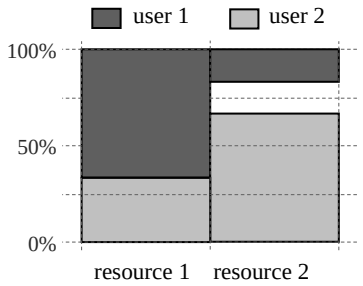

Figure 1: Example with two users with demand vectors $\langle 4, 1\rangle$ and $\langle 1, 2\rangle$. Given 12 units of two resources, Dominant Resource Fairness allocates the first user 2 tasks and the second user 4 tasks. Both users have dominant share $\frac{2}{3}$.

We next describe a simple model of a datacenter to facilitate our exposition of fairness.

2.1 Model

The system consists of n *users* $u_1, \cdots, u_n$, and m *resources* $R_1, \cdots, R_m$. A *resource vector* consists of $\langle r_1, \cdots, r_m\rangle$, where r_i denotes the total quantity of resource R_i. Users want to execute *tasks*. Each user i has a *demand vector* D_i, which describes its absolute per task[1] consumption, *e.g.,* $D_2 = \langle 3, 1, 2\rangle$ could signify that user u_2's tasks require 3 units of resource R_1, 1 units of resource R_2, etc. We refer to user u_i's demand of resource j as $D_{i,j}$, *i.e.,* $D_i = \langle D_{i,1}, \cdots, D_{i,m}\rangle$. Note that it is not necessary for users to specify their demand vectors to the scheduler in advance, rather the scheduler can in practise use users' actual usage of resources. We will often study the *positive demand vector* scenario, which is when $D_{i,j} > 0$ for every resource and user, as that is a common scenario in datacenters. A user is always given resources that are dimensioned as a multiple of that user's demand vector. An *allocation* is denoted $A = \langle a_1, \cdots, a_n\rangle$, and signifies that user u_i gets to run a_i tasks, dimensioned according to its demand vector D_i. Each user wants to run d_i tasks in total. It is, however, often easier to understand scheduling with infinite task demand, *i.e.,* $d_i = \infty$, which is henceforth assumed unless otherwise stated. To ease the analysis, we often use divisible resources, assuming that a fraction of a resource, such as CPU, can be allocated to a user. Since resources are often indivisible, we will approximate indivisible resources in our algorithms.

2.2 Dominant Resource Fairness

We describe DRF more formally here. Each user i has a *dominant share* x_i, which is i's share of its dominant resource, *i.e.,* $x_i = \max_j\{s_{i,j}\}$, where $s_{i,j}$ is

[1]This is a simplifying assumption. If a user has different resource demands for different tasks, we can model it as different users, using weighted fairness, as described in Section 2.4.

user i's fractional share of resource j, where $s_{i,j} = a_i D_{i,j}/r_j$.

We say that an allocation A is *Pareto efficient* if it is not possible to find an allocation in which every user has at least as many tasks as in A and at least one user has more tasks than in A. It might be impossible to equalize all users' dominant resource allocation and satisfy Pareto efficiency. For example, consider the resource vector $\langle 10, 10\rangle$ and three users with demand vectors $\langle 1, 0\rangle$, $\langle 0, 1\rangle$, $\langle 0, 1\rangle$. Allocating $A = \langle 5, 5, 5\rangle$ would be Pareto inefficient, as we could increase a_1 to 10 without decreasing a_2 and a_3. Consequently, we define DRF as the allocation that results from the following simple algorithm:

> *Repeatedly allocate one task to the user with minimum dominant share, for whom there are enough resources to allocate another task.*

Thus, among the users that can be allocated a task, allocate a task to $\min_i \max_j s_{i,j}$. The resulting allocation is clearly Pareto efficient.

The dominant resource fairness of an allocation can be quantified using Jain's Fairness Index (JFI) [3]. JFI prescribes the fairness metric: $\frac{(\sum_i y_i)^2}{n \sum_i {y_i}^2}$, where n is the number of users, and y_i is their resource share of a *single* common resource. JFI can be extended to multiple resources by letting y_i be the dominant share of user i, *i.e.,* $y_i = x_i$. A maximum JFI of 1.0 corresponds to scheduling resources according to dominant resource fairness, as it implies that every user has the same amount of their dominant resource. Hence, dominant resource fairness can be seen as a greedy algorithm for maximizing JFI.

2.3 Max-Min Fairness

Dominant resource fairness is an adaptation of *max-min fairness* [1, pg 448] to datacenters. A DRF allocation is max-min fair in the sense that any increase in a user p's tasks will be at the expense of a decrease in another user q's task, where q had a smaller share of its dominant resource than p had of hers. To see this, note that dominant resource fairness is an approximation of *progressive filling* [1, pg 450], in which all users' usage of their dominant resource is increased at the same rate, while proportionally increasing their other resource consumption, until some resource is exhausted, in which case those users' allocation is finalized. This process is repeated recursively for remaining users, until no more task allocations are possible.

Many basic results from max-min fairness for networks can be transferred to DRF. In the networking context, the problem is defined as a number of sources that are sending traffic through a graph in which each edge has a capacity constraint. Consider the well established result that a max-min fair allocation is equivalent with an allocation in which every source has a bottleneck link that is saturated and for which that source has the highest rate. This result is transferable to DRF, given divisible resources. We say that a *user u_j has a bottleneck resource R_k* in an allocation if R_k is fully utilized, and for each user u_i using R_k, $x_j \geq x_i$.

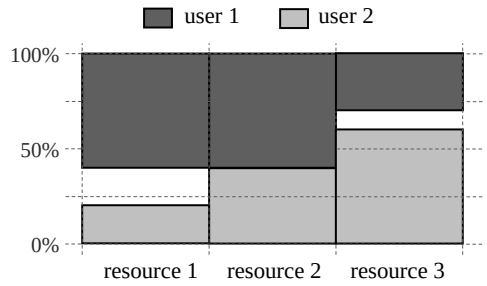

Figure 2: Example with two users with demand vectors $\langle 4, 4, 2\rangle$ and $\langle 1, 2, 3\rangle$. Given 20 units of three resources, Dominant Resource Fairness allocates the first user 3 tasks and the second user 4 tasks. Both users have dominant share 0.6.

Claim 2.1 (Max-min theorem) *A DRF allocation is equivalent to an allocation in which each user has a bottleneck resource.*

The proof of this mimics the original result [1] but replaces bottleneck resource with dominant resource.

For example, consider the resource vector $\langle 20, 20, 20\rangle$ and two users with demand vectors $\langle 4, 4, 2\rangle$, $\langle 1, 2, 3\rangle$. DRF gives 3 and 4 tasks to the respective users. The second resource will be fully utilized, whereas the other two are underutilized. Both users hence have R_2 as their bottleneck resource, with a dominant share of $x_1 = x_2 = 0.6$. As a side note, none of the dominant resources are fully utilized in this example.

2.4 Weighted Dominant Resource Fairness

We have so far assumed that resources are to be shared equally among the users. In practice, it is desirable to be able to weight the sharing. The motivation for this is that different users might represent different organizational entities, which might have contributed different amount of resources to the cluster or that there is a payment scheme in which different cluster owners pay for parts of the cluster. In either case, sharing could be weighted on a user basis, as well as on a resource basis. For example, some organization might have contributed with more memory resources, and should therefore have a higher share of the memory resources.

DRF supports weighted fair sharing similarly to Lottery Scheduling [10]. Every user i has a vector of weights of positive real numbers $w_{i,j}$, where $1 \leq j \leq m$ for m resources. The weight $w_{i,j}$ expresses that user i's fair proportion of resource j is $\frac{w_{i,j}}{\sum_k w_{k,j}}$. The definition of a dominant share for user i now changes to $x_i = \max_j\{\frac{s_{i,j}}{w_{i,j}}\}$. DRF now uses the new definition of a dominant share, x_i.

If $w_{i,j} = 1.0$ for all users and resources, weighted fair sharing reduces to equal fair sharing. Another use case is to assign user-wide weights. For example, if $w_{i,j} = k$ for all j, while $w_{k,j} = 1.0$ for all $k \neq i$ and all j, then user i will have

twice the proportion of all resources compared to every other user in the system. We have found that the latter use often suffices, as one would like to weight on a per-user level, and not on a per-resource level.

2.5 Desirable Fairness Properties

In this section we motivate and propose a number of desirable fairness properties. The next section compares different fairness policies using these properties.

Single Resource Fairness. In the case of a single resource, every allocation should be max-min fair. Hence, each user should be allocated $x_i = \min\{D_{i,1}, \alpha\}$, where α is chosen such that $\sum_i x_i = r_1$. In the case where each user has infinite demand on the single resource, this amounts to equally dividing the resource among the users. All policies in this paper satisfy this property.

Bottleneck Fairness. We call a resource a bottleneck resource if when all users' dominant resource coincides. If there is a bottleneck resource, bottleneck fairness requires that the bottleneck resource is allocated according to max-min fairness.

Share Guarantee. [2] In the general setting, with multiple resources and possibly heterogeneous demands, it is desirable to give some minimal guarantees. In particular, share guarantee requires each user to receive $\frac{1}{n}$ fraction of at least one of her resources. We believe this property to be important, as it can informally be interpreted as:

> *Each user will get at least as much resources as it would get by running its own cluster.*

Note that, assuming infinite task demand, the share guarantee implies bottleneck fairness. If share guarantee is satisfied, surely every user has $\frac{1}{n}$ of its dominant resource. If all users' dominant resources coincide, then every user must have exactly $\frac{1}{n}$ of the bottleneck resource, ensuring bottleneck fairness.

Population Monotonicity. If a user is removed, and her resources are relinquished, then the resulting allocation should not make any user worse off. This property is highly desirable, since if it is not satisfied, other users could be punished each time someone finishes a job.

Resource Monotonicity. If the available amount of some resource is increased, then the resulting allocation should not make any user worse off. This means that more resources can be added to the cluster without interrupting existing users. Since such upgrades are less frequent that users finishing jobs, this property is less crucial that population monotonicity.

[2] Share guarantee is known as *fair share* in microeconomic literature

Envy-freeness. A user *envies* another if it prefers that user's allocation to her own [2, 11, 9]. Thus, *envy-freeness* requires that no user ever envies another user in any allocation.

We would like to ensure that a user cannot receive more resources by providing a demand vector different from its actual demand. Envy-freeness suggests that this is hard to do, as a user will never envy another user, regardless of what demand vector that other user provided.

3 Alternative Fairness Policies

In this section we present other fair scheduling policies and compare them with each other.

3.1 Asset Scheduling

An perhaps very intuitive scheduling policy is to account for all resources that a user uses. We refer to this as *asset scheduling*. Informally, the goal is to schedule users to equalize every user's sum of resource usage. This might seem natural since the usage of a chunk of a resource can be equated with a fixed cost, which then implies that all users would be given resources for the same budget. This can be extended by using different costs for the different resources to account for resource importance or actual costs.

Asset scheduling assigns to each user u_i a *share sum* $x_i = \sum_j a_{i,j}$. It then applies max-min fairness by repeatedly allocating one task to the user with minimum share sum, for whom there are enough resources to allocate another task.

Claim 3.1 *Asset scheduling violates the share guarantee and bottleneck fairness.*

The following counter examples show this (see Figure 3). Consider the resource vector $\langle 30, 30\rangle$ and two users with demand vectors $D_1 = \langle 1, 3\rangle$, and $D_2 = \langle 1, 1\rangle$. Asset scheduling will allocate the first user 6 tasks and the second user 12 tasks, *i.e.,* $\langle 6, 12\rangle$. The first user will use $\langle 6, 18\rangle$ resources, while the second will use $\langle 12, 12\rangle$. While each user uses a total of 24 resources, the second user has got less than half (15) of both resources, violating the share guarantee. We believe that this could be considered unfair, making the second user inclined to buy a separate cluster of dimension $\langle 15, 15\rangle$, using it all by itself.

Asset scheduling also violates bottleneck fairness. Consider a 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 R_1 a bottleneck. Asset scheduling will allocate each user 3 tasks, only giving the first user $\frac{3}{7}$ of R_1, violating bottleneck fairness.

3.2 Dominant Resource Fairness

In the following we assuming divisible resources.

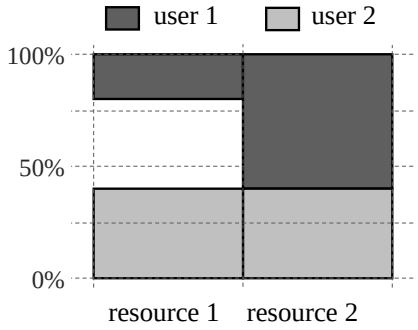

Figure 3: Example with two users with demand vectors $\langle 1, 3\rangle$ and $\langle 1, 1\rangle$. Given 30 of two resources, Asset scheduling gives the allocation $\langle 6, 12\rangle$, leaving the second user with less than half of both resources.

Claim 3.2 *DRF satisfies the share guarantee and bottleneck fairness.*

To see this, consider dividing each resource into n parts and giving each user one part of every resource. A user fully uses its dominant resource part, and only uses an amount of the rest of the resources such that its resource consumption matches its demand vector. This allocation might not be Pareto efficient, but every user has the same dominant share, $x_i = \frac{1}{n}$. Finally, note that the progressive filling algorithm for DRF will always pass through this allocation. Hence, each user will always get $\frac{1}{n}$ of its dominant resource, satisfying the share guarantee. As previously mentioned, this also implies that DRF satisfies bottleneck fairness.

In the following, we assume positive demand vectors and infinite task demand.

Claim 3.3 *Given positive demands, DRF guarantees that every user gets identical allocation of their dominant resource,* i.e., *every DRF allocation ensures* $x_i = x_j$ *for all* i, j.

This can be observed by noting that 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.

Claim 3.4 *Given positive demands, DRF satisfies population monotonicity.*

Consider any DRF allocation. Positive demands imply that all users have the same saturated resource or resources. 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 u_i's dominant share below α would, due to Pareto efficiency, have to allocate another user u_j a dominant share of more than α.

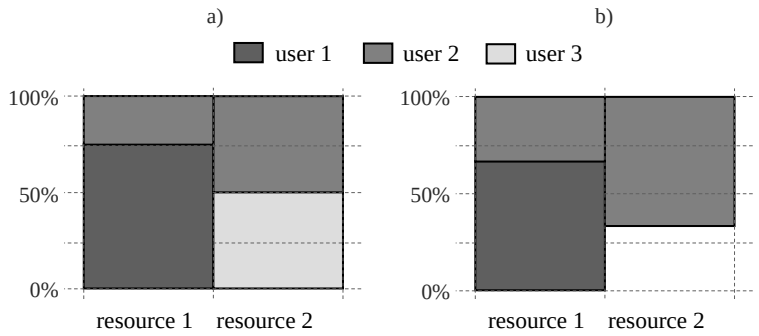

Figure 4: Example with three users with demand vectors $\langle 2, 0\rangle$, $\langle 1, 2\rangle$, $\langle 0, 2\rangle$. a) Given 24 units of three resources, DRF gives the allocation $\langle 9, 6, 6\rangle$. b) If the third user leaves, DRF gives the new allocation $\langle 8, 8\rangle$, violating population monotonicity.

The resulting allocation would violate max-min fairness, as it would be possible to increase u_i's dominant share by decreasing the allocation of u_j, who already has higher dominant share than u_i.

Things are, however, different without positive demand vectors.

Claim 3.5 *Without positive demands, DRF violates population monotonicity.*

Consider the resource vector $\langle 24, 24\rangle$ and three users with demand vectors $D_1 = \langle 2, 0\rangle$, $D_2 = \langle 1, 2\rangle$, and $D_3 = \langle 0, 2\rangle$ (see Figure 4). DRF will give the allocation $\langle 9, 6, 6\rangle$. Removing u_3 will give the DRF allocation $\langle 8, 8\rangle$, decreasing u_1's resources. This is because, before removing u_3, progressive filling will saturate R_2 first, giving all three users 6 tasks each. There is now available R_1 resources, which only u_1 can use, hence her allocation can be increased to 9 tasks. By removing u_3, R_2 does not become saturated, and hence both u_1 and u_2 demand it until it becomes saturated.

The consequence of the above is that any system, using DRF, that requires users to use some amount of every resource guarantees population monotonicity. This is sensible in a datacenter, as most applications use some amount of scheduling-relevant resources, such as CPU, memory, disk-space, and bandwidth. Consequently, no user would suddenly be amenable to task killing, because some other user finished running a task.

Claim 3.6 *DRF violates resource monotonicity.*

Consider the resource vector $\langle 12, 12\rangle$ and two users with demand $D_1 = \langle 2, 1\rangle$ and $D_2 = \langle 1, 2\rangle$ (see Figure 5). DRF gives the allocation $A_1 = \langle 4, 4\rangle$. If we double the first resource to $r_1 = 24$, DRF will give the new allocation $A_2 = \langle 6, 3\rangle$, which makes u_2 worse off. The reason for this is that the doubling of R_1 makes that resource more abundant. Hence, u_1's dominant share drops from $\frac{2}{3}$ to $\frac{1}{3}$

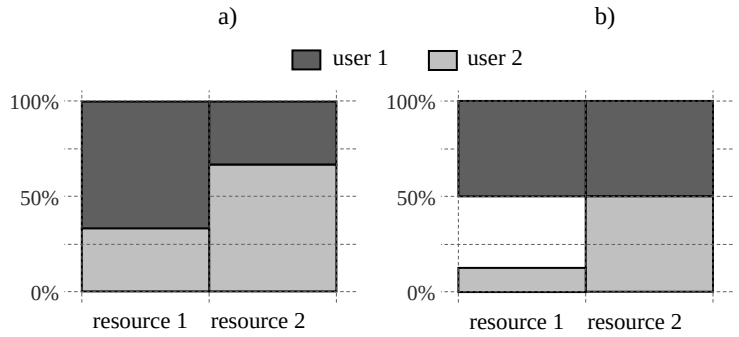

Figure 5: Example with two users with demand vectors $\langle 2, 1\rangle$, $\langle 1, 2\rangle$. a) Given 12 units of two resources, DRF gives the allocation $\langle 4, 4\rangle$. b) If the first resource is increased to 24, DRF gives the new allocation $\langle 6, 3\rangle$, violating resource monotonicity.

in A_1. The new DRF allocation A_2 equalizes the users' dominant shares by increasing u_1's dominant share to $\frac{1}{2}$, which requires decreasing u_2 the dominant share $\frac{1}{2}$. Consequently, any increase of a datacenter's resources might lead to some users' allocation going over their fair share, making them amenable to killing. This might be tolerable as such upgrades might be as infrequent as once a week.

We note that resource monotonicity is in some ways intrinsically at odds with having the share guarantee and Pareto efficiency. In particular, consider 2 resources and 3 users. User u_1 only wants R_1 and u_2 only wants R_2. User u_3 wants much more of R_1 than of R_2. Share guarantee requires than u_3 at least gets $\frac{1}{3}$ of R_1. Given an allocation that satisfies this, it is always possible to increase the total amount of R_1 such that u_3 has less than the share guarantee. To satisfy the share guarantee, u_3's share of R_1 has to increase, but that also requires an increase of its share of R_2. But by Pareto efficiency, R_2 is a saturated resource, so increasing u_3's share of R_2 will require decreasing u_2's share of R_2, thus violating resource monotonicity.

Claim 3.7 *Every DRF allocation is envy-free.*

This can be realized as follows. For a user u_i to envy another user u_j, u_j must have a strictly higher share of every resource than u_i, otherwise u_i cannot execute more tasks under u_j's allocation. But the max-min theorem implies that u_i must have a bottleneck resource R_k and none of the users using R_k have a higher dominant share than u_i. But this is impossible as u_j has a higher share of R_k than u_i and must therefore have a higher dominant share than u_i. Thus, u_i cannot envy any other user.

The envy-freeness of DRF shows that you will not be better off by demanding resources that you do not need. The following example shows that demanding unwanted resources can hurt your allocation, leading to a lower dominant share.

Consider a cluster with two resources, and 10 users, the first with demand vector $\langle 0, 1\rangle$ and the rest with demand vectors $\langle 1, 0$. The first user can be given all of R_1 and the rest of the users get $\frac{1}{9}$ of R_2 each. If instead changes its demand vector to $\langle 1, 1\rangle$, it can only be allocated $\frac{1}{10}$ of each resource and the rest of the users get $\frac{1}{10}$ of R_2. A similar example can be constructed with positive demands.

In practise, the situation can be exacerbated as resources in datacenters are typically partitioned in different physical machines, leading to fragmentation. Increasing ones demand artificially might lead to a situation in which, while there is enough resources on on a whole, there is not enough on any single machine to satisfy the new demand.

3.3 Fairness in Microeconomics

In Microeconomic Theory, there are two main methods to fairly divide resources: *Competitive Equilibrium from Equal Incomes* (CEEI) and *Egalitarian Equivalent* (EE) [5].

3.3.1 Competitive Equilibrium from Equal Incomes

The preferred method to fairly divide resources is based on the notion of envy-freeness [2, 11, 9]. As previously mentioned, a user envies another if it prefers that user's allocation to her own. The advantage of this definition is that it avoids making inter-personal comparisons of utility, *i.e.,* we only care if u_1 prefers u_2's allocation, not if u_2 enjoys her allocation more than u_1 enjoys his. Envy-freedom alone might be Pareto inefficient. For example, assuming heterogeneous demands, giving everyone $\frac{1}{n}$ of every resource is Pareto inefficient, as this leaves room for mutual trade which can improve both parties utility. Thus, an equitable allocation is one which is envy-free and Pareto efficient. Since many such allocations might exist, *competitive equilibrium from equal incomes* (CEEI) picks the one in which each user is given $\frac{1}{n}$ of every resource and then trades her resources with the other users in a perfect market. The outcome will be envy-free and Pareto efficient [9]. Note that the market has to satisfy the usual assumptions of perfect markets, such as price-taking[3] and market-clearance[4].

In absence of a real market, it is desirable to compute the CEEI. It turns out that if utilities are homogeneous, *i.e.,* $u(\alpha\, x) = \alpha\, u(x)$ for $\alpha > 0$, then the Nash bargaining solution [7, 5] from Game Theory gives the CEEI. 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. We refer to this as the Nash product from now on, and often maximize the logarithm of the product, *i.e.,* $\sum_i \log u_i(\alpha_i)$. In our model, the utility that a user gets from her allocation is simply its dominant share x_i. The allocation given by the Nash product is invariant under positive scaling, hence x_i and a_i give the same

[3] No single user can affect prices

[4] Prices are at an equilibrium, matching supply and demand, such that the market is cleared of the good.

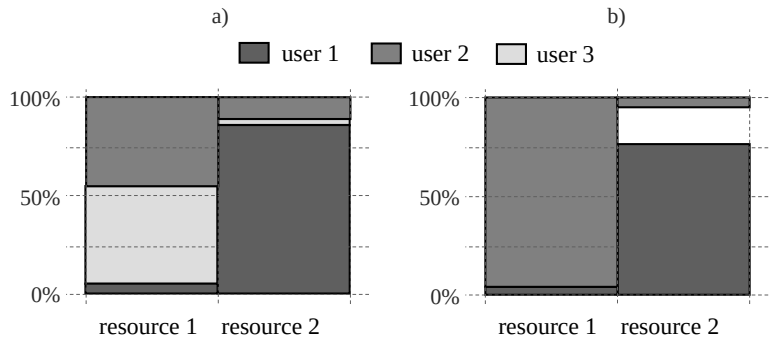

Figure 6: Example with three users with demand vectors $\langle 4, 1\rangle$, $\langle 1, 16\rangle$, $\langle 16, 1\rangle$. a) Given 1 units of two resources, DRF gives the allocation $\langle 11.28, 5.35, 3.09\rangle$. b) If the third user leaves, DRF gives the new allocation $\langle 23.8, 4.76\rangle$, violating population monotonicity.

allocation. Similarly, it will not matter if a user doubles its demand of every resource, as it will receive the same amount of resources.

Claim 3.8 *CEEI satisfies the share guarantee.*

This is obvious since every user starts with $\frac{1}{n}$ of every resource before starting to trade. Since it satisfies the share guarantee, it also satisfies bottleneck fairness.

Claim 3.9 *CEEI violates population monotonicity.*

This is the case even if positive demand vectors are assumed. Consider the 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 6). Numerically finding CEEI will then approximately give the allocation $A_1 = \langle 11.28, 5.35, 3.09\rangle$. If u_3 is removed and its resources relinquished, CEEI gives the new allocation $A_2 = \langle 23.80, 4.76\rangle$, which has made u_2 worse off than in A_1.

Claim 3.10 *CEEI violates resources monotonicity.*

The counterexample is the same as for the one for DRF, with identical resulting allocations (c.f. Section 3.2).

Finally, it can be useful to compare CEEI directly with DRF (see Figure 7). Consider the resource vector $\langle 100, 100\rangle$ and two users with demands $\langle 16, 1\rangle$ and $\langle 1, 2\rangle$. DRF gives the allocation $A_1 = \langle 4.16, 33.33\rangle$ while CEEI gives $A_2 = \langle 3.23, 48.39\rangle$. In A_1, user u_1 is using $\frac{1}{3}$ of R_1 while user u_2 is using $\frac{1}{3}$ of R_2. In A_2, u_1 using 51.61% of R_1 while u_2 using 96.78% of R_2. We think this could be considered unfair.

In the hypothetical limit it gets worse. Assume there are $m = n-1$ resources, with $r_i = 1$. Further, assume that u_1 has demand vector $\langle 1, 1, \cdots, 1\rangle$, while

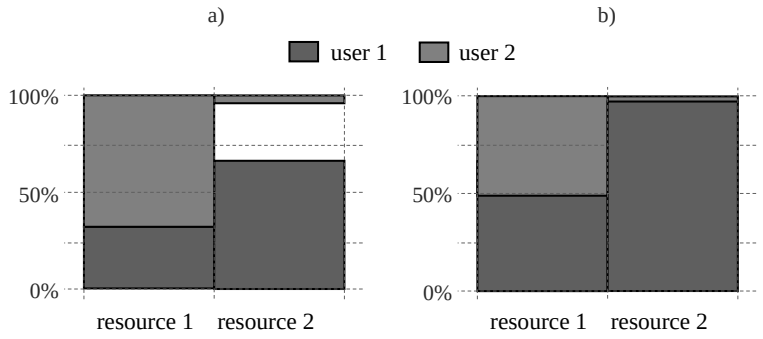

Figure 7: Example with two users with demand vectors $\langle 16, 1\rangle$, $\langle 1, 2\rangle$. a) Given 100 units of two resources, DRF gives the allocation $\langle 4.16, 33.33\rangle$. b) CEEI gives the allocation $\langle 3.23, 48.39\rangle$.

the other $n-1$ users have demand vectors $\langle 1, 0, 0, \cdots, 0\rangle$, $\langle 0, 1, 0, 0, \cdots, 0\rangle$, $\cdots$, $\langle 0, 0, \cdots, 1\rangle$. CEEI will allocate $a_1 = \frac{1}{n}$ while every other user $j \neq 1$ gets $a_j = \frac{n-1}{n}$. As $n \to \infty$, user u_1 starves.

3.3.2 Egalitarian Equivalent

Egalitarian equivalent was introduced since in some economic models CEEI does not exist [8]. Given homogeneous utilities, the EE allocation is given by the Kalai-Smorodinsky solution (KS) [4]. KS is based on the notion of a user's *maximum attainable utility* (MAU). It considers a user's utility minus its disagreement point and divides the difference by the user's MAU. KS attempts to equalize this ratio for all users.[5]. If multiple such allocations exist, it picks the one in which the ratio is maximized.

We apply the KS solution to our model by assuming that the utility functions are given by the dominant share of a user. Since the context is not game theoretic, a user's disagreement point will be zero. Given these constraints, the KS solution always coincides with the DRF solution.

Economists prefer CEEI over EE [11] because of a result [6] that proves that for every EE allocation there exists a utility profile which gives rise to envy. But as we showed earlier, every DRF allocation is envy-free. While the cited result shows that there *exist* utility functions that give rise to envy, our utility functions cannot be among them.

We finish our investigation by presenting the following table which compares the different schedulers.

[5]The disagreement point is used for coalition games in game theory. It is the point at which the user is as well off not cooperating with anyone.

Property	Scheduler		
	Asset	CEEI	DRF
Single Resource Fairness	×	×	×
Bottleneck Fairness		×	×
Share Guarantee		×	×
Population Monotonicity	×		×
Resource Monotonicity			
Envy-freeness	×	×	×

3.4 DRF Scheduling Oblivious to Demand Vectors

A DRF scheduler does not need to know the actual values of a user's demand vector. In fact, many users often do not know in advance their demand constraints. DRF only needs to observe the currently allocated resource shares of each user to be able to determine its dominant resource and dominant share. Based on that, the scheduler can pick the first schedulable user with lowest dominant share.

This leaves open the question of *how much* resources should be offered to the user scheduled by DRF. Assuming tasks that are fine-grained, as is indeed the case with many datacenter frameworks such as MapReduce and Dryad, the scheduler can offer all available resources to the picked user. The following scenario demonstrates this. At first, all datacenter resources are available. As soon as the first user enters, it is offered everything, and it accepts all resources. Now two other users join the datacenter. As tasks are short running in time and space (*i.e.,* resource requirements), the first user will soon finish a task that will immediately be offered to the second user. As soon as the next task finishes, it will be offered to the third user. This continues, such that soon all three users will reach a DRF equilibrium.

DRF's envy-free property incentivizes users to only use as much resources as they need. If they use more resources than they need, then their allocation might suffer. This also has the advantage that a user can switch its demand between resources. For example, assume a user is using a job that uses an algorithm that is CPU bound if it has less than a certain amount of memory, but it switches to an algorithm that is memory-bound when enough memory is available. Such a user can start by only accepting CPU resources. As soon as there is enough memory available, the user can switch to accepting memory resources. This user can make this transition smoothly, as its dominant share will be unaffected up to the point that its dominant resource changes from CPU to memory. Consequently, the user is disincentivized to hoard resources in advance, but knows that it can request them upon actual need.

4 Conclusion

We have introduced a notion of fairness, called Dominant Resource Fairness (DRF), for datacenter environments. DRF is an adaptation of max-min fairness from networking to datacenter environments. We listed a number of desirable

properties that a datacenter scheduler should have. Among them was the share guarantee, which requires each user to be given at least $\frac{1}{n}$ of one of its desired resources. We think that this property is important, since it informally translates to *every user gets as many resources as it would get if it ran a cluster by herself*. Another important property was population monotonicity, which roughly requires that the scheduler does not punish other users when some user finishes using resources. DRF satisfied the share guarantee and, under certain assumptions, population monotonicity. DRF is also envy-free, incentivizing users to correctly report their resource demand. DRF is more fair than other intuitive schedulers, as well as competing ones from microeconomic theory.

References

[1] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, second edition, 1992.

[2] D. Foley. Resource allocation and the public sector. *Yale Economic Essays*, 7(1):73–76, 1967.

[3] Raj Jain, Dah-Ming Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *CoRR*, cs.NI/9809099, 1998.

[4] Ehud Kalai and Meir Smorodinsky. Other Solutions to Nash's Bargaining Problem. *Econometrica*, 43(3):513–518, 1975.

[5] Herve Moulin. *Fair Division and Collective Welfare*. The MIT Press, 2004.

[6] Herve Moulin and William Thomson. Can everyone benefit from growth? : Two difficulties. *Journal of Mathematical Economics*, 17(4):339–345, September 1988.

[7] John Nash. The Bargaining Problem. *Econometrica*, 18(2):155–162, April 1950.

[8] Elisha A. Pazner and David Schmeidler. Egalitarian equivalent allocations: A new concept of economic equity. *Quarterly Journal of Economics*, 92:671–687, 1978.

[9] Hal Varian. Equity, envy, and efficiency. *Journal of Economic Theory*, 9(1):63–91, 1974.

[10] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: flexible proportional-share resource management. In *OSDI '94: Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, pages 1+, Berkeley, CA, USA, 1994. USENIX Association.

[11] H. Peyton Young. *Equity: in theory and practice*. Princeton University Press, 1994.