# Strategyproof Allocation of Multidimensional Tasks on Clusters

by

**Christos-Alexandros Psomas**

Master of Science Thesis

Graduate Program in Logic and Theory of Algorithms and Computation

University of Athens

Supervised by

**Aris Pagourtzis**

Committee

Aris Pagourtzis

Stathis Zachos

Evangelos Markakis

August 20th 2014

# Abstract

The present thesis focuses on the problem of fair resource allocation in a system containing multiple machines with multiple resources each. The users have heterogeneous demands and Leontief preferences, i.e. demand resources in *fixed* proportions. Resource allocation is a key issue in the design of cloud computing systems. Traditional solutions, like *max-min* fairness per resource don't work well in this multi resource setting. Furthermore, efficiency and fairness are not the only issues here; the designer must take into account the users' incentives.

In the past couple of years this problem has received a lot of attention from the algorithmic game theory community. We review some the most important results related to multi-resource allocation, starting from the work of Ghodsi et al ([7]) that studied the problem on a single machine setting with fractional tasks. We then move on to the indivisible tasks on a single machine case, studied by Parkes et al ([13]). Finally we discuss the work of Friedman et al ([4]) that studies the problem of executing indivisible, containerized tasks on a multiple machine setting.

# Περìληψη

Στην παροὺσα εργασìα μελετοὺμε το πρὸβλημα της δìκαιης κατανομὴς πὸρων σε ενα σὺστημα με πολλοὺς υπολογιστὲς με πολλοὺς πὸρους ο κὰθε ènας.Οι χρὴστες èχουν διαφορετικὲς απαιτὴσεις και Leontief προτιμὴσεις, δηλαδὴ απαιτοὺν πὸρους σε σταθεροὺς λὸγους. Η δìκαιη κατανομὴ πὸρων εìναι κεντρικὸ πρὸβλημα στην σχεδìαση συστημὰτων cloud computing . Παραδοσιακὲς λὺσεις ὸπως max-min fairness ανὰ πὸρο δεν δουλεὺουν ικανοποιητικὰ σε τὲτοια συστὴματα με πολλοὺς πὸρους. Επιπλèον, η αποδοτικὸτητα και η δικαιοσὺνη δεν εìναι τα μὸνα προβλὴματα. Ο σχεδιαστὴς πρèπει να λὰβει υπὸψην τα κìνητρα των χρηστὼν.

Τα τελευταìα χρὸνια αυτὸ το πρὸβλημα èχει τραβὴξει την προσοχὴ της κοινὸτητας της αλγοριθμικὴς θεωρìας παιγνìων. Θα μελετὴσουμε τα πιο σημαντικὰ αποτελèσματα σχετικὰ με την κατανομὴ πολυδιὰστατων πὸρων, ξεκινὼντας απο την δουλειὰ των Ghodsi et al ([7]) που μελὲτησε το πρὸβλημα σε συστὴματα με èναν υπολογιστὴ με κλασματικὲς διεργασìες. Συνεχìζουμε με διακριτὲς διεργασìες σε èναν υπολογιστὴ, περìπτωση που μελετὴθηκε απο τους Parkes et al ([13]) . Τèλος μελετοὺμε τη δουλειὰ των Friedman et al ([4]) που κοιτὰει το πρὸβλημα εκτèλεσης διακριτὼν διεργασιὼν σε συστὴματα με πολλοὺς υπολογιστὲς.

# Contents

# Acknowledgements

First and foremost I want to thank Aris Pagourtzis for his guidance and all the time he dedicated to helping me complete this thesis. My special thanks to the other members of the committee, Stathis Zachos and Vangelis Markakis. I would also like to thank all the faculty of the $\mu\Pi\lambda\forall$ graduate program for providing me with a great environment to learn and grow. Also many thanks to my colleagues and friends from $\mu\Pi\lambda\forall$ and CoreLab; I had a lot of fun with you guys.

I would like to thank my co-authors Eric Friedman and Ali Ghodsi for all their hard work. Many thanks to Christos Papadimitriou and all my friends and colleagues from Berkeley.

Last but not least I would like to thank my family. Without them none of this would have been possible.

# Chapter 1

# Introduction

## 1.1 Motivation and Background

The problem of allocating jobs in a cloud computing center is of broad practical and theoretical interest. For the former, cloud computing centers (CCCs), both internal (such as used by Twitter) and external (such as Amazon's EC2) are used for a large fraction of the world's computing and growing rapidly. In these systems, both servers and users are heterogeneous: modern datacenters are likely to be constructed from a variety of classes with different amounts resources (CPU, Memory etc) per class. In addition to server heterogeneity cloud computing centers, there is extreme user diversity as well: some users might want to execute CPU intensive tasks, other might want memory intensive tasks etc. In these systems traditional max-min fairness approaches and single resource abstractions fall short. Moreover, one has to take into account the fact that the users are strategic: our task is to design *mechanisms*, not *algorithms*. Anecdotal evidence from cloud operators indicates that *strategyproofness* is important, since attempting to manipulate schedulers is very common. For example, a big search company provided dedicated machines for jobs only if the users could guarantee high utilization. Soon the company found that users would add infinite loops to their code to artificially inflate utilization.

The introduction of the dominant resource fairness ([7]) protocol has spurred much work on strategyproof allocation for Leontief Economies ([13], [2], [11]) leading to important insights and unexpected connections between computer

science ([9]) and economics ([5]). DRF has been extended to be used inside routers ([6]) as well as large organizations with hierarchies ([1]). In addition, it has been deployed in production, running on thousands of nodes at Twitter in the Mesos resource manager ([8]).

The basic model underlying the analysis of DRF is an extremely simplified setting in which there is a single large machine and job allocations can be fractional. While this has led to a widely used mechanism in CCCs, many issues, such as the discreteness of jobs and the distributed computing environment have been dealt with in ad-hoc manners. In this thesis we review the main results in this area, starting from the seminal paper of Ghodsi et al ([7]). We move on to various extensions of DRF, focusing on the single machine with indivisible jobs case by Parkes et al ([13]). Then we consider a model which captures both the discreteness of jobs and the multiplicity of machines, first discussed in Friedman et al ([4]).

## 1.2 Thesis Outline

In chapter 2 we formally define some notation that we use through this thesis, and present the main mechanism: **Dominant Resource Fairness**. This mechanism is defined in a single machine setting with divisible tasks. We prove that it satisfies pareto optimality, strategyproofness, sharing incentives (aka individual rationality) and envy freeness. This chapter is based on the work of Ghodsi et al ([7]).

In chapter 3 we discuss some extensions of the Dominant Resource Fairness mechanism, with our main focus being it's extension to indivisible tasks by Parkes et al ([13]). In this setting we prove that the main properties are incompatible if we insist on deterministic mechanisms. We also present some positive results (the Sequential Min-Max algorithm) that are possible if we give up on strategyproofness.

In chapter 4 we introduce the main setting of this thesis: multiple machines with indivisible tasks, that are allocated as containers. We present our main (randomized) mechanism *Containerized DRF* (CDRF), and prove it satisfies all the desired properties in expectation. This chapter and the next are based on the work of Friedman et al ([4]).

In chapter 5 we consider mechanisms that approximate CDRF, and their performance in an ex-post sense. We prove that (deterministic) allocations that are close to the expected CDRD allocation approximately satisfy strategyproofness and sharing incentives.

# Chapter 2

# Dominant Resource Fairness

## 2.1 Preliminaries

We will define some basic notation that will used throughout this thesis. A cluster has $m$ machines, $n$ users and $p$ resources. Each machine $j$ has a vector of resources $r_j \in \Re_+^p$, i.e. $r_{jk} \geq 0$ of resource $k$. We will refer to the cluster, the vector of machine vectors, as $r$. User $i$ has true demand $d_i \in \Re_+^p$ for each job, where $d_{ik}$ is the amount of resource $k$ that she needs to execute a task. We will say that $d = (d_1, d_2, \ldots, d_n)$ is a demand profile.

An allocation **A** is a distribution of resources between the users. Given allocation $A$, user's $i$ utility for that allocation $u_i(A)$ will be the number of tasks she can execute in the cluster in that allocation. This number can either be real or integer, depending on the model we're working on. A mechanism **M** is a mapping from demand profiles to allocations. We want to design mechanisms that satisfy a number of properties. We will define these properties informally to avoid introducing notation that we will not use later on.

A mechanism satisfies the *Sharing Incentives* (also known as individual rationality) if each user's utility is at least her utility for the allocation that gives $\frac{1}{n}$ of the cluster to each user. A mechanism is *Strategyproof* if no user can increase her utility by misreporting her demand vector. A mechanism is *Pareto Optimal* if the output allocation is efficient: there does not exist an allocation which is strictly better for (at least) one user and every other user is at least as well off. A mechanism is *Envy-Free* if each user prefers her own bundle to the bundle received by any other agent.

In the rest of the chapter we define the **Dominant Resource Fairness** mechanism (DRF), defined by Ghodsi et al ([7]) for the single machine case ($m = 1$) and divisible tasks.

## 2.2 The DRF mechanism

Consider a single machine with 9 CPU's and 18 GB's of RAM, and two users. User 1 wants to execute tasks that require $(1\,CPU, 4\,GB)$ each, and user 2 tasks that require $(3\,CPU, 1\,GB)$ each. In other words, user 1 has demand vector $d_1 = (1, 4)$, and user 2 has demand vector $d_2 = (3, 1)$. The cluster consists of a single machine with resource vector $(9, 18)$.

The **Dominant Resource Fairness** (DRF) mechanism works as follows: The *dominant resource* of a user is the resource for which the agent's tasks require the largest fraction of total availability. In the example, user's 1 dominant resource is RAM, since she demands $\frac{2}{9}$ of the total RAM available per task, while she only needs $\frac{1}{9}$ of the total CPU per task. Similarly, user's 2 dominant resource is CPU. The *dominant share* of a user is simply the fraction of her dominant resource she got allocated. DRF tries to maximize the number of allocated tasks, subject to equal dominant shares.

Back to the example, if $x$ is the number of tasks allocated by DRF to user 1, and $y$ the number of tasks allocated to user 2, then the total number of resources allocated are $(x+3y)$ CPU's and $(4x+y)$ GB's RAM. The dominant shares are $\frac{4x}{18} = \frac{2x}{9}$ and $\frac{3y}{9}$ respectively. The DRF allocation is the solution to the following maximization problem:

$$\begin{aligned} \max \quad & (x, y) \\ \text{s.t.} \quad & x + 3y \leq 9 \\ & 4x + y \leq 18 \\ & \tfrac{2}{9}x = \tfrac{3}{9}y \end{aligned}$$

The solution is $x = 3$ and $y = 2$. Thus, user 1 gets allocated $(3CPU, 12GB)$ and user 2 gets allocated $(6\,CPU, 2\,GB)$. See Figure 2.1.

It is useful to note that DRF need not always equalize dominant shares: if for example a resource gets exhausted, users that do not need it can still continue getting higher shares. We will later give alternate interpretations of DRF that will make this clear.

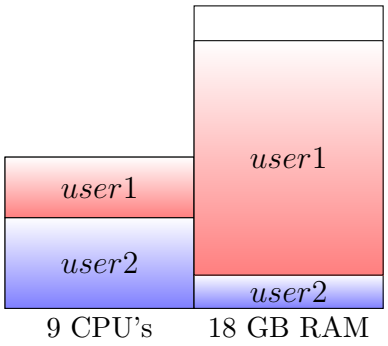


Figure 2.1: Resources per user in the example: User 1 gets $(3\,CPU, 12\,GB)$ and user 2 gets $(6\,CPU, 2\,GB)$

## Implementing DRF

The DRF allocation can be computed very easily in an iterative fashion: at every step pick the user with the lowest dominant share and give her enough resources to increase her dominant share by some fixed amount $\epsilon$ (if there are enough resources to do so).

Consider the previous example, and for simplicity at each step the algorithm will allocate a whole task. See Table 2.1. DRF first picks user 2 to execute a task. As a result the shares become $(\,{}^{3}/_{9}, {}^{1}/_{18}\,)$ and the dominant share is ${}^{1}/_{3}$. Next, DRF picks user 1 (dominant share is 0) and executes on of her tasks, changing her dominant share from 0 to ${}^{2}/_{9}$. In the next step user 1 is selected again $(\,{}^{2}/_{9} < {}^{1}/_{3}\,)$, and so on, until it is no longer possible to execute new tasks.

## Properties of DRF

Ghodsi et al ([7]) show that DRF satisfies all the properties we discussed:

**Theorem 1.** *The DRF allocation mechanism satisfies sharing incentives, strategyproofness, pareto optimality and envy-freeness.*

In addition DRF satisfies other "nice-to-have" properties: *Single resource fairness* (for a single resource, the solution is equivalent to the max-min solution), *Bottleneck fairness* (if the dominant resource is the same for all users, then the solution is equivalent to the max-min solution for that resource), *Population monotonicity* (if a user leaves the system none of the allocations of the

| Schedule | User 1 | | User 2 | | CPU | RAM |
|---|---|---|---|---|---|---|
| | resources | dom. share | resources | dom. share | | |
| User 2 | $(0, 0)$ | **0** | $(3/9, 1/18)$ | $1/3$ | $3/9$ | $1/18$ |
| User 1 | $(1/9, 4/18)$ | $\mathbf{2/9}$ | $(3/9, 1/18)$ | $1/3$ | $4/9$ | $5/18$ |
| User 1 | $(2/9, 8/18)$ | $4/9$ | $(3/9, 1/18)$ | $\mathbf{1/3}$ | $5/9$ | $9/18$ |
| User 2 | $(2/9, 8/18)$ | $\mathbf{4/9}$ | $(6/9, 2/18)$ | $2/3$ | $8/9$ | $10/18$ |
| User 1 | $(3/9, 12/18)$ | $\mathbf{2/3}$ | $(6/9, 2/18)$ | $\mathbf{2/3}$ | 1 | $14/18$ |

Table 2.1: Progressive filling implementation

other users decreases). The authors of [7] also compare DRF with other fair allocation policies, namely Asset Fairness and Competitive equilibrium from equal incomes.

We will now prove Theorem 1. The interested reader can refer to [7] for the comparison with Asset Fairness and CEEI.

**Lemma 2.** *DRF is Strategyproof.*

*Proof.* Assume that some user $i$ can increase her dominant share by submitting a demand vector $d_i' \neq d_i$. Let $x_{i,k}$ and $x'_{i,k}$ be the amount of resource $k$ user $i$ gets, under progressive filling, when submitting the truth and lying respectively. For user $i$ to be better off we need $x'_{i,k} > x_{i,k}$ for every resource $k$ with non-zero demand. Let $r$ be the first resource that becomes saturated for user $i$ when submitting $d_i$. If no other user is allocated resource $r$ we have a contradiction (there is no allocation that increases user $i$'s utility). Thus, there are other users that are allocated resource $r$. Let $t$ be the step that progressive filling saturates $r$ under $d_i$, and $t'$ the step it saturates $r$ under $d_i'$. Since the dominant share in $x'$ is higher, it must be that $t' > t$. Thus, $i$ under $d_i'$ does not saturate any resources before step $t'$, and thus doesn't affect other users allocations before step $t'$. So, under $d_i'$, since at every step the increase in dominant share is the same, a user $j$ has $x_{j,r}$ of resource $r$ at time $t$, same as under $d_i$. Thus, it isn't possible that $x'_{i,r} > x_{i,r}$.

□

We're going to user without proof the following Lemma:

**Lemma 3.** *Every user in a DRF allocation has at least one saturated resource.*

**Lemma 4.** *DRF is Pareto Optimal.*

*Proof.* Assume that user $i$ can increase her dominant share $s_i$ without decreasing the dominant share of anyone else. According to Lemma 3, user $i$ has at least one saturated resource. If no other user is using the saturated resource we have a contradiction. If other users are using the saturated resource then increasing the allocation of $i$ would result in decreasing the allocation of some other user $j$. Since under progressive filling the resources allocated by any user are proportional to her demand vector, decreasing the allocation of any resource of user $j$ would also decrease her dominant share. This contradicts our hypothesis and thus proves the result. □

**Lemma 5.** *DRF satisfies Sharing Incentives.*

*Proof.* Assume resource $k$ is the first resource being saturated by progressive filling. Let $i$ be the user allocating the largest share on resource $k$ and let $t_{i,k}$ denote her share on $k$. Since resource $k$ is saturated we have $t_{i,k} \geq \frac{1}{n}$. Furthermore, by the definition of the dominant share, we have that $s_i \geq t_{i,k} \geq \frac{1}{n}$. 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, and thus DRF satisfies sharing incentives. □

**Lemma 6.** *DRF is Envy Free.*

*Proof.* Assume by contradiction that some user $i$ envies the allocation of another user $j$. For that to happen, user $j$ must have a strictly higher share of every resource that $i$ wants, otherwise she cannot execute more tasks under $j$'s allocation. This means that $j$'s dominant share is strictly larger than $i$'s dominant share. Since every resource allocated to $i$ is also allocated to $j$ (otherwise $i$ wouldn't envy $j$), user $j$ cannot reach its saturated resource *after* $i$, i.e. $t_j \leq t_i$, where $t_k$ is the step that user's $k$ 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 user's $i$ dominant share can only increase, violating the hypothesis.E □

# Chapter 3

# Indivisible Tasks on a Single Machine

Even though the Dominant Resource Fairness mechanism was a significant step in fair allocation of multidimensional resources under Leontief preferences, the setting considered was too restricting: the cluster had a single machine, and fractional tasks where allowed (half the demand vector results in utility half). The paper quickly attracted attention from the algorithmic game theory community, leading to many extensions. We will briefly discuss some of them, and then focus on the results of Parkes et al ([13]) about executing indivisible tasks on a single machine.

## 3.1 Extensions of DRF

### Alternative interpretations

A nice alternative way of thinking about DRF, due to Parkes et al ([13]), is the following: let $D_{ir}$ be the ratio between the demand of user $i$ for resource $r$ and the availability of that resource. In the example of Figure 2.1 $D_{11} = \frac{1}{9}$, $D_{12} = \frac{4}{18}$, $D_{21} = \frac{3}{9}$ and $D_{22} = \frac{1}{18}$. For all users $i$ and resources $r$ let $\hat{d}_{ir} = D_{ir} / \max_{r'} D_{ir'}$; the normalized demands. In the example $\hat{d}_{11} = \frac{1}{2}$, $\hat{d}_{12} = 1$, $\hat{d}_{21} = 1$ and $\hat{d}_{22} = \frac{1}{6}$.

Let $x$ be the dominant share of each agent (remember that the dominant shares are equal). Thus, every user is allocated an $(x \cdot \hat{d}_{ir})$-fraction of resource $r$. Now, instead of writing a linear program to compute $x$, one can simply find it by computing:

$$x = \frac{1}{\max_r \sum_i \hat{d}_{ir}}$$

In the example $x = \frac{1}{(\frac{1}{2}+1)} = \frac{2}{3}$. Thus, user 1 gets $\frac{2}{3} \cdot \frac{1}{2} = \frac{1}{3}$ of the available CPU, $\frac{2}{3}$ of the available RAM, and user 2 gets $\frac{2}{3}$ of the total CPU and $\frac{1}{9}$ of the total RAM.

### Extended DRF

The way we defined DRF, it doesn't allow for zero demands, or weighted users (i.e. users with different priorities). Parkes et al ([13]) show that with a slight modification of DRF we can address these issues without losing any properties.

Assume that each user $i$ has a publicly known weight $w_{ir}$ for resource $r$ which reflects the user's endowment of that resource. We assume w.l.o.g. that for all resources $r$, $\sum_{users\, i} w_{ir} = 1$. Pareto Optimality and Strategyproofness do not need to be redefined, but Envy-Freeness and Sharing incentives do. Sharing Incentives now means that each user $i$ receives as much utility as she would from the allocation that assigns her a $w_{ir}$-fraction of each resource. Envy-Freeness requires that user $i$ does not envy user $j$ when the allocation of $j$ is scaled by $\frac{w_{ir}}{w_{jr}}$ on each resource $r$.

Now, we'll need to make some adjustment to the definition of a dominant resource. Let $r_i^*$ be the *weighted dominant resource* of user $i$, defined by

$r_i^* = \arg\min_{r:d_{ir}>0} {}^{w_{ir}}/_{\hat{d}_{ir}}$. Next, let $\rho_i = {}^{w_{ir_i^*}}/_{\hat{d}_{ir_i^*}}$ be the ratio of weight to demand on the weighted dominant resource of user $i$. The **Extended DRF** mechanism proceeds in rounds: let $s_{rt}$ be the surplus fraction of resource $r$ left unallocated at the beginning of round $t$. The fraction of resource $r$ allocated to $i$ in round $t$ is denoted $A_{ir}^{(t)}$. Weights and demands do not change during the execution of the algorithm, but an agent's dominant resource might. In pseudocode Extended DRF is given in 1.

**ALGORITHM 1:** Extended DRF

**Input**: Demand profile $d$, weights $w$

**Output**: A feasible allocation $A$

$t \leftarrow 1$ , $\forall r, s_{r1} \leftarrow 1$ , $S_1 \leftarrow N$ ($N$ is the set of users)
**while** $S_t \neq 0$ **do**
  $x_t \leftarrow \min_r \left( \frac{s_{rt}}{\sum_{i \in S_t} \rho_i \cdot \hat{d_{ir}}} \right)$
  $\forall i \in S_t$, resources $r$, $A_{ir}^{(t)} \leftarrow x_t \cdot \rho_i \cdot \hat{d_{ir}}$
  $\forall i \in N \setminus S_t$, resources $r$, $A_{ir}^{(t)} \leftarrow 0$
  $\forall$ resource $r$, $s_{r,t+1} \leftarrow s_{rt} - \sum_{i \in S_t} A_{ir}^{(t)}$
  $t \leftarrow t + 1$
  $S_t \leftarrow \{i \in S_{t-1} | \forall r, \hat{d_{ir}} > 0 \Rightarrow s_{rt} > 0\}$
**end while**
$\forall$ users $i$, resources $r$, $A_{ir} = \sum_{k=1}^{t-1} A_{ir}^{(k)}$

Extended DRF deviates from the unweighted mechanism in a couple of ways. First, instead of $x = \frac{1}{\max_r \sum_i \hat{d_{ir}}}$, we have a different $x_t$ in each round $t$. Thus, each agent $i$ is now allocated an $(x_t \cdot \rho_i \cdot \hat{d_{ir}})$ fraction of resource $r$ in round $t$.

We can show that Extended DRF satisfies the main properties, and even is *Group Strategyproof*:

**Theorem 7.** *Extended DRF satisfies Pareto optimalality, Envy-freeness, Sharing Incentives and Group Strategyproofness.*

We will not prove this theorem. The reader can refer to [13] for a proof.

## DRFH

The last extension of DRF we'll discuss is the one by Wang et al ([16]). The authors consider a setting with multiple machines and fractional tasks. In a multi-machine setting computing the DRF allocation in each machine separately results in very undesirable outcomes. For example, consider a cluster of

two machines with two resources each $r_1 = (10, 100)$ and $r_2 = (100, 10)$. Now, assume we have two users with demands $d_1 = (1, 10)$ and $d_2 = (10, 1)$. Doing DRF in each machine separately ignores the dovetailing of user 1 in machine 1 and user 2 in machine 2. Thus, we have to look for alternatives.

The authors propose a mechanism called **DRFH** (DRF in Heterogeneous servers). The idea behind DRF is to achieve the max-min fair allocation of the *global dominant resources*. The global dominant resource of user $i$ is her dominant resource on the aggregate resource vector $\sum_{machines\, j} r_j$. So, if $A_{il}$ is the allocation of user $i$ in machine $l$, let $G_{il}(A_{il})$ be the fraction of global dominant resources user $i$ gets on machine $l$, her *global dominant share* on machine $l$. Thus, a user's overall *global dominant share* is

$$G_i(A_i) = \sum_{machines\, l} G_{il}(A_{il})$$

DRFH aims to maximize the minimum global dominant share among all users, subject to the resource constraints per machine:

$$\max_{allocation\, A} \min_i G_i(A_i)$$

$$s.t. \sum_i A_{ilr} \leq r_{lr}, \forall\, machines\, l, resources\, r$$

DRFH satisfies a number of important properties, but as we'll see later, does not satisfy *sharing incentives* as well as other important robustness criteria.

## 3.2 Indivisible Tasks and Impossibility results

Parkes et al ([13]) remove the divisible tasks assumption and consider a single machine setting with indivisible tasks. This time, user's $i$ utility for a bundle of resources is a step function, increasing for every additional task she can execute given that bundle. In this setting, as we will see, deterministic mechanisms do not perform as well.

As one moves to this setting with indivisibilities, DRF is **not** Pareto Optimal anymore. To see this most clearly consider a single machine cluster, with one resource of size 1, and two users with demands $d_1 = \frac{1}{10}$ and $d_2 = \frac{2}{5}$. DRF would allocate the resource half and half, resulting in utilities 5 and 1 for users 1 and 2 respectively. Observe though that allocating $\frac{3}{5}$ to user 1 and $\frac{2}{5}$ to user 2, leaves user 2's utility unchanged, but increases user 1's utility by 1.

Unfortunately, this is not just an issue with DRF. One cannot hope for a good deterministic algorithm in this setting:

**Theorem 8.** *No mechanism satisfies Pareto Optimality, Strategyproofness and Sharing Incentives simultaneously.*

*Proof.* Consider a system with a resource of size 1 and two users with demands $d_1 = d_2 = \frac{1}{2} + \epsilon$, for some small $\epsilon > 0$. Assume w.l.o.g. that a mechanism that satisfies all 3 properties allocates to user 1. If user 2 reports $d_2' = \frac{1}{2}$ then sharing incentives dictates that she should get half the resource. But, since the remaining half cannot be used by user 1 and it can be used by user 2, Pareto-optimality dictates that user 2 gets that as well. So, user 2 has a profitable deviation and the mechanism is not strategyproof. □

Moreover, Pareto Optimality is inherently incompatible with Envy-Freeness: when each agent needs all the available resources to execute a task, the only Pareto Optimal allocations are the ones that give all the resources to some user $i$. These allocations of course are not envy free. Notice also that if we give up on Pareto Optimality it is trivial to satisfy the other 3 properties with the following mechanism: give $\frac{1}{n}$ of each resource to each user.

Parkes et al ([13]) define a relaxed notion of envy-freeness, that they call EF1 (envy free up to one bundle). A mechanism satisfies EF1 if for every submitted demand profile $d$ it outputs an allocation $A$, such that for all users

$i, j,\ u_i(A_i) \geq u_i(A_j - d_i)$, where $A_k$ is the vector of resources user $k$ was allocated. In other words, a mechanism satisfies EF1 if each user $i$ does not envy user $j$ if one instance of the task of user $i$ is removed from the allocation of $j$.

Again, we have that the main properties are incompatible with each other:

**Theorem 9.** *No mechanism satisfies Pareto Optimality, EF1 and Strategyproofness simultaneously.*

*Proof.* Consider a setting with one resource of size 1 and two users with demands $d_1 = d_2 = \frac{1}{3}$. The only feasible allocations that are EF1 and Pareto Optimal allocate $\frac{2}{3}$ of the resource to one user and $\frac{1}{3}$ to the other user. Assume without loss of generality that user 1 gets two tasks. If user 2 reports $d_2 = \frac{1}{6}$, then the only EF1 and PO allocation gives her $\frac{2}{3}$ and $\frac{1}{3}$ to user 1, thus violating strategyproofness.

□

So, to summarize, we have that Strategyproofness + Sharing Incentives + EF1 is trivial, but if we add Pareto Optimality and insist on Strategyproofness we run into impossibilities. If we give up Strategyproofness though, as we'll see next, there does exist a deterministic mechanism that satisfies Pareto Optimality, EF1 and Sharing Incentives: **Sequential Min-Max**.

## 3.3 Sequential Min-Max

To describe Sequential Min-Max we will need to define some additional notation: given an allocation $A$ let $MaxDom(A) = \max_{users\, i} \max resources\, rA_{ir}$, where $A_{jk}$ is the amount of resource $k$ user $j$ gets. $MaxDom(A)$ is the maximum dominant share a user gets. Also, let $A \uparrow i$ be the allocation obtained by starting from allocation $A$ and giving user $i$ another bundle $d_i$.

**Sequential Min-Max** (Algorithm 2) allocates one bundle at each step, to the user that minimizes the maximum dominant share after the allocation. Note that maximizing the minimum dominant share instead, which is arguably a more intuitive approach, does not achieve the same properties.

**ALGORITHM 2:** Sequential Min-Max

**Input**: Demand profile $d$

**Output**: A feasible allocation $A$

$k \leftarrow 1$ , $A^0 \leftarrow 0$ , $T_1 \leftarrow n$
**while** $T_k \neq 0$ **do**
  $M_k \leftarrow \{i \in T_k | \forall j \in T_k,\ MaxDom(A^{k-1} \uparrow i) \leq MaxDom(A^{k-1} \uparrow j)\}$
  $i \leftarrow$ any agent in $M_k$
  $A^k \leftarrow A^{k-1} \uparrow i$
  $T_{k+1} \leftarrow \{i \in T_k | A^k \uparrow i$ feasible $\}$
  $k \leftarrow k + 1$
**end while**
Return $A^{k-1}$

**Theorem 10.** *Sequential Min-Max satisfies Pareto Optimality, Sharing Incentives and EF1*

We will not prove this theorem here. The interested reader can refer to [13] for the proof of this theorem.

To summarize, in this section we moved away from the divisible tasks assumption, and we immediately hit impossibilities for our main properties and deterministic mechanisms. In the next chapter (Chapter 4) we will consider a setting with multiple machines and indivisible tasks, but allow randomized mechanisms to evade these kinds of impossibilities.

# Chapter 4

# Containerized DRF

## 4.1 Containers

In this chapter we move to our main, most general setting: multiple machine cluster with indivisible tasks. Before the formal model we present an illustrating example (Figure 4.1):

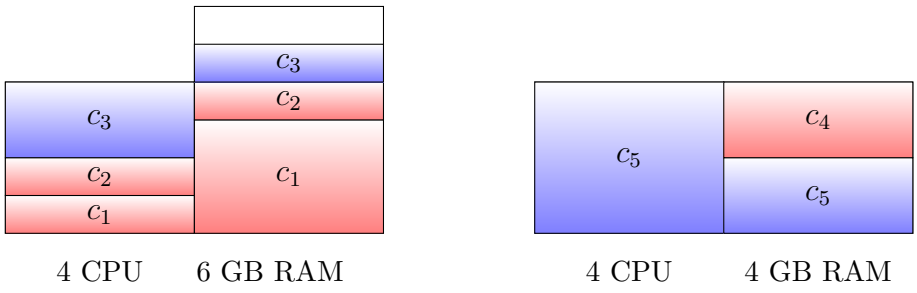


Figure 4.1: Example of an allocation in a containerized setting with machines $r_1 = (4, 6)$ and $r_2 = (4, 4)$, and users $d_1 = (1, 3)$ and $d_2 = (2, 1)$. User 1 gets containers $c_1 = (1, 3)$, $c_2 = (1, 1)$ and $c_4 = (0, 2)$. User 2 gets containers $c_3 = (2, 1)$ and $c_5 = (4, 2)$

**Example:** Consider a cluster of two machines, with two resources each: machine 1 has 4 CPU's and 6 GB RAM, while machine 2 has 4 CPU's and 4 GB RAM. Two users want to execute their tasks in this cluster. User 1 wants to execute tasks that require 1 CPU and 3 GB's of RAM each. User 2 wants to execute tasks that require 2 CPU's and 1 GB RAM. In other words, the users have demands $d_1 = (1, 3)$ and $d_2 = (2, 1)$ respectively. In a cluster like this we

will assume that each user gets allocated a set of *containers*; each container is a bundle of resources. A user can execute in each container she gets as many tasks as she can "fit". Her total utility is simply the total number of tasks she can execute in the cluster. So, say user 1 is allocated containers $c_1 = (1, 3)$ and $c_2 = (1, 1)$ in machine 1, and container $c_4 = (0, 2)$ in machine 2. User 2 gets allocated container $c_3 = (2, 1)$ in machine 1 and $c_5 = (4, 2)$ in machine 2. User 1 can only use $c_1$; $c_2$ and $c_4$ are too small. Moreover, she can't combine these two containers (this holds even if the containers were in the same machine). Thus, her utility is 1. Similarly, user 2 has utility 3: she can fit one task in $c_3$ and two tasks in $c_5$.

## Utilities in a containerized setting

If there was a single machine and user $i$ was allocated a bundle $x \in \Re_+^p$ of resources, then in a model with divisible tasks she could run $\min_k(x_k/d_{ik})$ jobs, while in a model without fractional tasks she could only run $\lfloor \min_k(x_k/d_{ik}) \rfloor$ jobs. On the other hand, in a setting with many machines users shouldn't be able to combine resources across machines. The key here is that the system allocates containers. The $s$'th container $c_s \in \Re_+^p$ is a bundle of resources on a single machine, i.e. $c_{sk}$ is the amount of resource $k$ in the container $s$. In a containerized model, using container $c_s$ user $i$ can run $\lfloor \min_k(c_{sk}/d_{ik}) \rfloor$ jobs. Combining resources across containers is not possible.

An **allocation** $A =< C, M, P >$ consists of a set $C$ of containers with a machine function $M(s) \in [m]$ and a user function $P(s) \in [n]$. $M(s)$ is the machine on which container $s$ is located and $P(s)$ the user who gets container $s$. For an allocation $A$, the number of jobs that user $i$ can execute is

$$u_i(A, d_i) = \sum_{\{s|P(s)=i\}} \lfloor \min_k(c_{sk}/d_{ik}) \rfloor$$

and note that the floor function is inside the sum. This number is exactly a user's utility. Observe that we have Leontief preferences: users require resources in fixed proportions. We will often write $u_i(A)$ when the $d_i$ is implicit.

An allocation is feasible, if

$$\forall j \in [m], k \in [p] : \sum_{\{s|M(s)=j\}} c_{sk} \leq r_{jk}$$

We denote by $F(d)$ the feasible region, that is the set of all feasible allocations under submitted demand profile $d$. We will also use $F_j(d)$ for the feasible region of machine $j$.

A **mechanism** $A(d, r)$ is a function that takes as input a demand profile $d$ and a cluster $r$ and outputs an allocation. In much of the following we will consider a fixed set of machines and resources, so will often simply write $A(d)$ when $r$ is implicit. The mechanisms we will design will only allocate containers equal to demands, so the usage of containers will be implicit.

Our mechanisms work directly with the feasible region $F(d)$, so it is useful to define the following:

**Definition 11.** *The Minkowski sum of sets $A_1, A_2, \ldots, A_k$ is the set*

$$B = \{\sum_{i=1}^{k} x_i : x_i \in A_i\}$$

Observe that $F(d)$ is the Minkowski sum of the $F_j(d)$'s. We will later use the fact that the Minkowski sum of convex hulls is the convex hull of Minkowski sums.

## Properties of mechanisms

Our goal is to design mechanisms that satisfy our key properties: *Pareto Optimality*, *Strategyproofness* and *Sharing Incentives*. The definitions of these properties are informally as before, except from sharing incentives. To define this property, we first define user's $i$ *stand alone allocation* in the cluster to be the number of jobs user $i$ could run if she had the entire system for herself

$$sa_i(d, r) = \sum_{j=1}^{m} \lfloor \min_k (r_{jk}/d_{ik}) \rfloor$$

We will usually write $sa_i$ to relax notation. A mechanism satisfies *Sharing Incentives* if every user's utility for the output allocation is at least her *fair share* $\lfloor sa_i/n \rfloor$.

We will not consider Envy Freeness, since it is not a very natural property in a setting that allocates multidimensional containers. On the other hand, we will consider two additional "nice-to-have" properties:*population monotonicity* and *independence of dummy machines*. A mechanism $M$ is *population monotonic* if no user is harmed (receives fewer jobs) when some other user leaves

the system. The last property we want our mechanisms to satisfy is independence of dummy machines, *IDM*, where a dummy machine is one on which no user can execute any job: A mechanism satisfies *IDM* if adding or removing dummy machines does not change the allocation. IDM is a simple proxy for robustness, since if a mechanism fails IDM then it will fail a variety of other important robustness considerations.

## The importance of containers

The importance of containers is that they allow the underlying placement of jobs on machines to be invisible to the users. Many cloud providers (e.g. Amazon.com) indeed hide from users whether two container instances are co-located or not. The user is therefore given the illusion of owning a dedicated machine that is isolated from the other instances, even though she is running on a container on a machine that has other containers. There are many reasons for this. First, it enables container migration. Second, not revealing co-location avoids applications that rely on co-location, limiting the flexibility of the provider (once one customer relies on this, you have to support it forever or break their applications). Finally, cloud providers often do not want to reveal too much information about the exact technology they are using as that is a competitive advantage as well as a potential vulnerability that attackers can exploit. Thus, unlike previous papers on DRF, in our model the system directly allocates containers (the term is borrowed from the Linux literature, c.f. [12]), which are isolated bundles of resources. This differs from the model in [13] as users cannot combine bundles. For example, if a user needs 2 units of a certain resource to run a job, but for strategic reasons only requests 1 unit per job, then even if she gets allocated 2 jobs on the same machine she is unable to run her job, even though she does have a total of 2 units of the resource. Of course the user could execute separate tasks in each container and have them use the network to coordinate as if they were a single task, but this would have performance overheads (communication), as well as cost overheads (parallelizing the code). Furthermore, a task might have the consumption vector (3 CPU, 4 GB memory), and it cannot simply separate it into one (3 CPU, 3 GB memory) and one (0 CPU, 1 GB memory) tasks.

## 4.2 Possibilities and Impossibilities

### Limitations of Deterministic Mechanisms

As in the single machine case, when we demand integer allocations no mechanism can satisfy sharing incentives, Pareto optimality, and strategyproofness simultaneously. For the model without containers and a single machine, this was proven by the following example in [13] (Theorem 8): consider a system with a resource of size 1 and two users with demands $d_1 = d_2 = \frac{1}{2} + \epsilon$. Assume w.l.o.g. that a mechanism that satisfies all 3 properties allocates to user 1. If user 2 reports $d_2' = \frac{1}{2}$ then sharing incentives dictates that she should get half the resource. But, since the remaining half cannot be used by user 1 and it can be used by user 2, Pareto-optimality dictates that user 2 gets that as well. So, user 2 has a profitable deviation and the mechanism is not strategyproof.

However, this example does not prove impossibility in our model. To see this consider a mechanism that allocates containers equal to demand vectors. Then user 2 would get containers of size $\frac{1}{2}$ when she reported $d_2'$. But she cannot execute any jobs in these containers, and she cannot combine them, so her utility when misreporting would be zero. Nonetheless, a slightly more complicated example does work as seen below:

**Theorem 12.** *No deterministic mechanism can satisfy sharing incentives, Pareto optimality and strategyproofness simultaneously, even for one machine, two resources and two users.*

*Proof.* Assume such a mechanism $M$ exists and consider a cluster with one machine and two resources $r = (1, 1)$ and two users with demands $d_1 = (0.25, 0.1)$ and $d_2 = (0.1, 0.25)$. Since $M$ satisfies sharing incentives then each user should be allowed to execute at least 2 tasks. A Pareto optimal mechanism should allocate one more job for either user 1 or user 2. Without loss of generality assume that $M$ chooses the first option and allocates containers which result to utility 3 for user 1 and utility 2 for user 2.

Now, examine what happens when user 2 deviates with $d_2' = (\frac{1}{6}, 0.25)$. Again, sharing incentives requires $M$ to allocate containers that give utility at least 2 to each user. This will use at least $(\frac{5}{6}, 0.7)$ of the available resources, leaving $(\frac{1}{6}, 0.3)$ available, which is enough for user 2 to schedule a task, but not

user 1. Pareto optimality dictates that the remainder should be allocated to user 2. Since $d_2$ "fits" in $d_2'$, user 2's utility strictly increases, thus she has an incentive not to report her true demand vector and so, $M$ is not strategyproof, a contradiction. □

## SI allocations always exist

Even though all three main properties are impossible to satisfy at the same time, it is very easy to construct a mechanism that satisfies only Pareto optimality and strategyproofness. For example, a serial dictatorship, in which we choose a fixed order of users and allow each user, in order, to allocate as many jobs as possible. However, it is not clear if there always exists an allocation that satisfies sharing incentives for every cluster. In this subsection we will prove that such an allocation always exists by showing the correctness of a moving knife-like algorithm ([3]). In addition to providing a useful result for multiple machine settings, this will also be necessary for the correctness proof of the main randomized mechanism.

Recall the moving knife procedure for allocating a divisible cake to $n$ players, when each player's value for the whole cake is 1: the referee moves a knife from left to right until some player $i$ calls "cut". Then the referee cuts at the point where the knife was and allocates to player $i$ everything on the left of the cut. If each player calls "cut" when she thinks the amount of cake on the left of the knife is worth $\frac{1}{n}$ the allocation is fair: everybody gets utility at least $\frac{1}{n}$.

Now, in our setting, given an arbitrary ordering of the machines, imagine a water filling procedure for every user $i$ that works as follows: starting from the first machine and following the ordering, user $i$ progressively fills the cluster using a constant fraction of the resources. Every time she has "filled" enough to be able to allocate her fair share $\lfloor \frac{sa_i}{n} \rfloor$ she puts a mark $\mu_{i,k}$. So, the marks are such that, if she were allocated everything between two consecutive marks, she would get exactly her fair share, with the exception maybe of the last mark, where she would possibly get more if allocated everything from that mark until the end. There are $n-1$ such marks.

We will write $\mu_{i,k} = (f, j)$ to denote that the $k$-th mark of user $i$ is in machine $j$, when she reached an $f$ fraction of that machine. Notice that this

doesn't mean that she needs an $f$ fraction of machine $j$ to get her fair share, since two consecutive marks can be many machines apart. We will say that $\mu_{i,k} \leq \mu_{j,k}$ if $\mu_{i,k}$ is a mark in a machine earlier in the ordering, or in the case that both marks are on the same machine, $\mu_{i,k}$ uses a smaller (or equal) fraction.

We are now ready to state the algorithm:

**ALGORITHM 3:** Moving Knife Algorithm

**Input**: Cluster $r = (r_1, \ldots, r_m)$ and demand profile $d$

**Output**: A feasible allocation $A$

```
Pick an arbitrary ordering of the machines
For each user i compute n − 1 marks μ_{i,k}
S = set of users
μ = (0, 1) ( mark the beginning of the first machine )
while |S| > 1 do
    Pick user j ∈ S with smallest μ_{j,n−|S|+1} mark
    Allocate to j everything from μ until μ_{j,n−|S|+1}
    μ = μ_{j,n−|S|+1}
    S = S \ {j}
end while
Allocate the rest to the last user
```

*Example:* Consider the cluster from the previous section ( Figure 4.1 ), with an extra user $d_3 = (1, 2)$, and an extra machine with 4 CPU's and 2 GB RAM. User 1, with $d_1 = (1, 3)$, would be able to execute 3 jobs if alone in the system: 2 jobs in machine 1 and 1 job in machine 2, so her fair share is $\frac{3}{n} = 1$. She will put $n - 1 = 2$ marks: $\mu_{1,1} = (\frac{1}{2}, 1)$ and $\mu_{1,2} = (1, 1)$. Similarly user 2, with $sa_2 = 6$, would have marks $\mu_{2,1} = (1, 1)$ and $\mu_{2,2} = (1, 2)$. User 3, with $sa_3 = 6$ as well, would have $\mu_{3,1} = (\frac{2}{3}, 1)$ and $\mu_{3,2} = (\frac{1}{2}, 2)$, because she can get her fair share with $\frac{1}{3}$ of machine 1 and $\frac{1}{2}$ of machine 2. Figure 4.2 shows the execution of the algorithm in this example.

**Theorem 13.** *The mechanism defined by the moving knife Algorithm 3 satisfies sharing incentives.*

*Proof.* Assume some user $i$ did not get her fair share of the system. She was allocated some part of the system either in the first step, in some step $k > 1$ or in the last step. In the first and last step cases we have an immediate contradiction from the description of the algorithm; the user was definitely allocated her fair share. The only interesting case is if she got picked in some

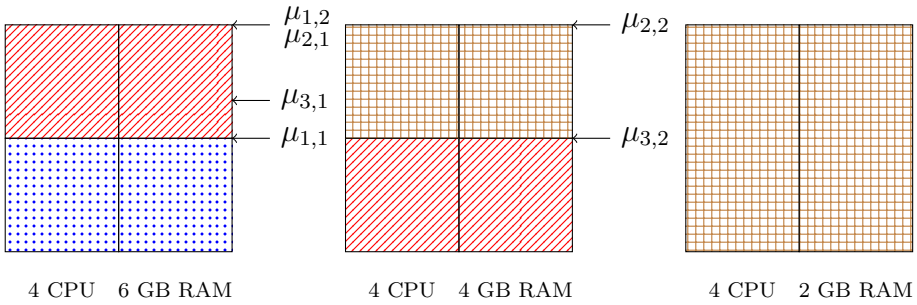


Figure 4.2: Marks in a cluster with $r_1 = (4, 6), r_2 = (4, 4)$ and $r_3 = (4, 2)$, and users with demands $d_1 = (1, 3), d_2 = (2, 1)$ and $d_3 = (1, 2)$. After the execution of algorithm 3, the dotted area shows the fraction user 1 gets, the lined area shows the fraction user 3 gets, and the grid the fraction user 2 gets

step $k > 1$. In that case, we know that she was not picked in step $k-1$ because there was some other user $j$ that had a mark $\mu_{j,k-1} \leq \mu_{i,k-1}$. User $i$ then got allocated everything between $\mu_{j,k-1}$ and $\mu_{i,k}$ which is more than everything between $\mu_{i,k-1}$ and $\mu_{i,k}$, which is by definition her fair share. □

## 4.3 The CDRF mechanism

As we showed in Theorem 12, one cannot exactly satisfy the basic properties with a deterministic mechanism. However, as we shall now show there does exist a randomized mechanism, which is the natural extension of DRF, that satisfies all of the desired properties on average (in an ex-ante sense). Since our mechanisms allocate containers equal to demands we will relax notation and refer to allocations simply as vectors $y \in \mathbb{N}^n$, where $y_i$ is user's $i$ utility, i.e. the number of jobs she can execute in that allocation. On the other hand, when talking about non integral solutions, allocations are vectors in $\mathbb{R}^n_+$. This distinction should be clear from the context.

We will restrict our randomized mechanisms to be a random mixture of a finite set of deterministic mechanisms. Our goal is to construct a randomized mechanism which satisfies all of the desirable properties on average before the randomization is revealed, i.e. ex-ante. We note that these do not necessarily satisfy the properties ex-post, that is when the actual allocation is initialized, and discuss this issue later.

For most properties ex-post is a stronger guarantee than ex-ante. Notice though that, somewhat surprisingly, ex-ante Pareto optimality implies ex-post Pareto optimality, but not the converse. This happens because in order to achieve ex-ante Pareto optimality one must make sure that the average allocation is not dominated. To see this distinction, consider a randomized serial dictatorship: pick a random order on the players and allocate as many jobs as possible to the first player in the order, then to the second player etc. This mechanism is fair, strategyproof and ex-post Pareto optimal, but not ex-ante Pareto optimal. For example, in a cluster with two users with demands $d_1 = (10, 1)$ and $d_2 = (1, 10)$ and two machines with $r_1 = (100, 10)$ and $r_2 = (10, 100)$ the randomized serial dictatorship yields allocations $(11, 0)$ and $(0, 11)$ with equal probability combining for an ex-ante allocation of $(5.5, 5.5)$. This is a dramatic loss of efficiency considering that allocation $(10, 10)$ is also feasible. Incidentally, $(10, 10)$ will be the output of our randomized mechanism with probability 1.

First, before we start describing our mechanism, we note that one can reinterpret DRF. In the case of a single machine with divisible jobs, notice that

the dominant share $s_i = x_{ik^*}/r_{k^*}$ is equal to the ratio of allocated jobs to user $i$ divided by the number of potential jobs for user $i$ if she had the entire machine to herself. This equivalence allows us to connect DRF to the Kalai-Smorodinsky bargaining solution ([10]). To see this most clearly, recall that the total number of jobs which can be allocated to user $i$ is $sa_i$, which in the single machine with divisible jobs is simply $\min_k \frac{r_k}{d_{ik}} = \frac{r_{k^*}}{d_{ik^*}}$. The generalized Kalai-Smorodinski solution, if translated in this setting, is the weighted max-min allocation of jobs with weight vector $(1/sa_1, \ldots, 1/sa_n)$ over the feasible region. The DRF allocation is exactly this weighted max-min vector of jobs.

One could extend this definition to our container based model; however, if done directly, this fails because the set of feasible allocations $F(d)$ is not convex, a fact crucial to the analysis. To resolve this, we convexify the feasible region to $CH(F(d))$, the convex hull of set $F(d)$, and compute the max-min $\frac{1}{sa_i}-$weighted vector of jobs over $CH(F(d))$. A geometrical interpretation of this solution is this: the weighted max-min is just the intersection of the Kalai-Smorodinski line, the line connecting the origin to the $(sa_1, \ldots, sa_n)$ point, with $CH(F(d))$.

As we will show, this procedure, which we denote Containerized DRF (CDRF), preserves the properties of DRF in this multiple-machine with indivisible jobs setting. In addition, one can directly interpret this mechanism as a randomized mechanism, since the resulting allocation is a convex combination of allocations in $F(d)$, by the definition of a convex hull. Moreover these allocations are Pareto optimal. In pseudocode, CDRF is given as Algorithm 4.

**ALGORITHM 4:** Containerized-DRF

**Input:** Demand profile $d$ and cluster $r$

**Output:** A feasible allocation $z \in \mathbb{N}^n$.

1: Compute the allocation $CDRF(d, r) = (k_1, k_2, \ldots, k_n) \in \Re^n$ which is the max-min $\frac{1}{sa_i}-$weighted vector of jobs over $CH(F(d))$

2: Compute $n$ allocations $z_1, \ldots, z_n \in F(d)$ such that $CDRF(d, r)$ is their convex combination, i.e. $\exists b_1, \ldots, b_n \in \Re_+$ s.t.

$$\sum_{j=1}^{n} b_j z_j = CDRF(d, r) \text{ and } \sum_{j=1}^{n} b_j = 1$$

3: Output allocation $z_j$ with probability $b_j$

**Example:** The feasible region is $n$ dimensional, so, for simplicity, we use the example from Figure 4.1: two machines $r_1 = (4, 6)$ and $r_2 = (4, 4)$ and users

with demands $d_1 = (1, 3)$ and $d_2 = (2, 1)$. See Figure 4.3. The feasible region is the set of integer points and the colored area is it's convex hull. The set of Pareto optimal points of the feasible region are $\{(3, 1), (2, 2), (1, 3), (0, 4)\}$. The stand alone shares are $sa_1 = 4$ and $sa_2 = 3$. The intersection of the Kalai-Smorodinski line, connecting the origin with the $(sa_1, sa_2)$ point, is the point that maximizes the minimum $\frac{u_i}{sa_i}$ over the colored area. In this example that point is $CDRF(d, r) = (\frac{12}{7}, \frac{16}{7})$. This can be written as a convex combination of $(1, 3)$ and $(2, 2)$ with coefficients $\frac{2}{7}$ and $\frac{5}{7}$. Thus, CDRF will output the allocation $(1, 3)$ with probability $\frac{2}{7}$, and the allocation $(2, 2)$ with probability $\frac{5}{7}$.

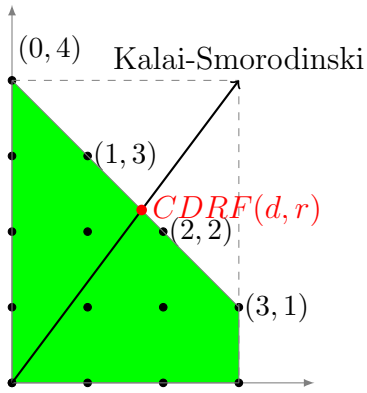


Figure 4.3: CDRF in a cluster with $r_1 = (4, 6)$ and $r_2 = (4, 4)$, and users with demands $d_1 = (1, 3)$ and $d_2 = (2, 1)$. $CDRF(d, r)$ is the weighted max-min allocation, i.e. $CDRF(d, r)_i$ is the expected utility of user $i$ in CDRF. In higher dimensions $CDRF(d, r)$ is not necessarily the intersection of $CH(F(d))$ and this "Kalai-Smorodinski" line, as one needs to use the lexicographic max-min.

Let $CDRF(d, r) \in \Re^n$ be the weighted max-min allocation for demand profile $d$; the Kalai-Smorodinski solution. Observe that the expected number of containers user $i$ gets is exactly $CDRF_i(d, r)$. If $i$ reports her true demand that $CDRF_i(d, r)$ is also her utility. Since the region we're maximizing over is convex, it immediately follows that Containerized-DRF satisfies several key properties.

**Theorem 14.** *If users are expected utility maximizers then CDRF satisfies ex-ante sharing incentive, ex-ante Pareto optimality, ex-ante population monotonicity and independence of dummy machines.*

*Proof.* Pareto optimality is ex-ante satisfied since the expected allocation is on the convex hull of the feasible region. Sharing incentives is similarly satisfied ex-ante since the Kalai-Smorodinski solution always dominates the $(\lfloor sa_1/n \rfloor, \ldots, \lfloor sa_n/n \rfloor)$ allocation, and we know from Theorem 13 that this allocation is always feasible. Population monotonicity also follows directly since when a user leaves the system, the feasible region can only expand. The independence of dummy machines property holds since our mechanism works directly with the feasible region: if the feasible region is not changed, then the allocation is not changed. □

CDRF is also strategy proof in expectation, although the proof is somewhat more complex.

**Theorem 15.** *If users are expected utility maximizers then CDRF is ex-ante strategyproof.*

*Proof.* Let $d'$ be the demand profile where the $i$-th user reports some $d'_i$ and every other user $l$ truthfully reports $d_l$, and $d$ the demand profile where everyone is truthful. Also, let $f_i(d'_i, d_i)$ be the number of tasks user $i$ can execute in a container of size $d'_i$. Remember that $CDRF_i(d', r)$ is the expected number of containers user $i$ gets in profile $d'$. So, her expected utility can be written $CDRF_i(d', r)\ f_i(d'_i, d_i)$. We have to show that this value is maximum for $d'_i = d_i$.

First observe that by definition reporting a vector $d'_i$ with $d'_{ir} < d_{ir}$ for some resource $r$ gives zero utility, since $f_i(d'_i, d_i)$ is zero. So, we only have to consider deviations $d'_i$ that over demand resources.

Let $p_k = \lfloor \frac{d'_{ik}}{d_{ik}} \rfloor$. If all $p_k$ are not equal, the largest can be reduced without any loss in utility: $f_i(d'_i, d_i)$ will be the same, $CH(F(d))$ and $sa_i$ can only increase, thus $CDRF_i(d', r)$ can only increase. Using the exact same argument we can see that $p_k = \frac{d'_{ik}}{d_{ik}}$, that is $d'_i$ is a better deviation for our mechanism when it is an integer multiple of $d_i$. So, w.l.o.g. we can assume that all $p_k$'s are equal to some integer $p$, and that $d'_{ik} = pd_{ik}$.

All that's left to show is that the best such $p$ is $p = 1$. Let's examine what happens when the demand of user $i$ changes from $d_i$ to $d'_i = p\ d_i$: $sa_i$ becomes at least $p$ times smaller, since it is harder to allocate big tasks of size $d'_i$ than it is to allocate tasks of size $d_i$. For the same reason, $CH(F(d'))$ is a subset

of $CH(F(d))$ with the $i$-th dimension rescaled by $p$. So, $CDRF_i(d', r) \leq \frac{CDRF_i(d,r)}{p}$, while $f_i(d'_i, d_i) = p$, for any $p \geq 1$. Thus there is no profitable deviation.

□

## DRFH in a containerized setting

One could directly extend DRFH (3.1) to our containerized model, by maximizing the global dominant resource shares; however, this mechanism fails to satisfy some basic properties:

*Example:* Consider one machine with resource vector $r_1 = (15, 15)$ and two users with demand vectors $d_1 = (1, \frac{1}{2})$ and $d_2 = (\frac{1}{2}, 1)$. Under DRFH, 1's global dominant resource is resource 1, since in order to execute a single task she needs a $\frac{1}{15}$ fraction of resource 1 and a $\frac{1}{30}$ fraction of resource 2. Similarly, the global dominant resource of user 2 is resource 2. DRFH allocates 10 tasks to each user. Note that each user's stand alone share is $sa_i = 15$ so this allocation satisfies the resource sharing property. However, if we add a second machine with $r_2 = (16, 0)$ then even though this machine is unable to run a single job for either user, it changes the allocation significantly, giving 12 jobs to user 1 and 6 to user 2. This happens because user 1's dominant resource is now resource 2.

This example shows that DRFH need not satisfy the sharing incentives property and that it also lacks the important independence of dummy machines property, as the addition of dummy machine changes the allocation significantly.

# Chapter 5

# Ex-post Guarantees

## 5.1 Approximate CDRF mechanisms

In certain situations, the randomized mechanism might be directly applicable. For example, if individual job times are short then in a randomized mechanism the fluctuations in users' job shares would average out quickly. However, one potential problem with this mechanism in practice is the possibility of large fluctuations in allocations which may rely too heavily on the assumption that users maximize expected utility. For example, consider the case with $r_j = (1)$ for $1 \leq j \leq m$ and $d_1 = d_2 = (1)$. The randomized CDRF mechanism could choose to randomize between two allocations, the first gives all the machines to user 1 and the second gives all the machines to user 2. Thus, while the average allocation for each user is $m/2$ jobs, the fluctuations are huge.

In the following, we present several mechanisms which help resolve this issue. They produce allocations which not only satisfy the properties in the expectation/ex-ante sense, but also approximately satisfy these properties ex-post, i.e. after the actual allocations are realized. The key idea is that if we can always find an allocation that is close to the CDRF allocation, then the mechanism that produces that allocation will "inherit" approximate ex-post bounds for the properties satisfied by CDRF. We formalize this as follows:

**Definition 16.** *A mechanism $M$ is an $\epsilon$-approximate CDRF mechanism if for all demand profiles $d$ and clusters $r$,*

$$\frac{|M_i(d,r) - CDRF_i(d,r)|}{CDRF_i(d,r)} \leq \epsilon$$

*for all users $i$, and $M$ is ex-ante Pareto Optimal.*

In order to simplify the notation, in this section we let $M_i(d, r)$ be the number of jobs allocated to user $i$ and require that the mechanism allocates containers of size $d_i$. Thus the explicit discussion of containers will not be necessary.

It is straightforward to show that such a mechanism approximately satisfies the ex-post versions of our properties, where we define these directly in terms of the fractional changes to each user. For example, $\epsilon$-approximate ex-post sharing incentive implies that the actual allocation

$$M_i(d, r) \geq (1 - \epsilon)\lfloor \frac{sa_i(d, r)}{n} \rfloor$$

and similarly for $\epsilon$-approximate ex-post population monotonicity and $\epsilon$-approximate IDM. For $\epsilon$-approximate ex-post strategyproofness we require that no user can gain a fractional increase in jobs of $\epsilon$ from a deviation.

**Theorem 17.** *If mechanism $M$ is an $\epsilon$-approximate CDRF mechanism then it satisfies:*
*1) $\epsilon$-approximate ex-post sharing incentives,*
*2) ex-ante Pareto Optimality,*
*3) $\epsilon$-approximate ex-post population monotonicity,*
*4) $\epsilon$-approximate IDM,*
*5) $3\epsilon$-approximate ex-post strategyproofness.*

*Proof.* Parts 1,3 and 4 follow immediately from the definition of "$\epsilon$-approximate ex-post" versions. Part 2 is immediate from the assumption that $M$ is ex-ante Pareto Optimal. Part 5 follows from the strategyproofness of CDRF: Consider a deviation for a single user from $d_i$ to $d'_i$, with $d$ and $d'$ the respective demand profiles. Assuming that $d_{ik} \leq d'_{ik} < 2d_{ik}$, for all resources $k$, we can see that

$$\frac{|u_i(M_i(d'), d'_i) - u_i(CDRF_i(d'), d'_i)|}{u_i(CDRF_i(d'), d'_i)} \leq \epsilon$$

as is

$$\frac{|u_i(M_i(d), d_i) - u_i(CDRF_i(d), d_i)|}{u_i(CDRF_i(d), d_i)} \leq \epsilon$$

by the definition of an $\epsilon$-approximate mechanism. Next, by strategyproofness of CDRF, we see that $u_i(CDRF_i(d'), d_i) - u_i(CDRF_i(d), d_i) \leq 0$ and by the

assumption on $d'_i$, $u_i(CDRF_i(d'), d'_i) = u_i(CDRF_i(d'), d_i)$, since user $i$ can only fit one job in a container of size $d'_i$. Similarly $u_i(M_i(d'), d') = u_i(M_i(d'), d)$. Also, from the definition of $\epsilon$-approximate CDRF we can get $\frac{u_i(M_i(d),d_i)}{1-\epsilon} \geq u_i(CDRF_i(d), d_i)$. Combining these facts we get:

$$\frac{u_i(M_i(d'), d_i) - u_i(M_i(d), d_i)}{u_i(CDRF_i(d), d_i)} \leq 2\epsilon$$

which implies

$$\frac{u_i(M_i(d'), d_i) - u_i(M_i(d), d_i)}{u_i(M_i(d), d_i)} \leq \frac{2\epsilon}{1-\epsilon} < 3\epsilon$$

which is the definition of $3\epsilon$ approximate ex-post strategyproofness. In the case where $d'_i$ does not satisfy our assumption, we use the same techniques as in the proof of theorem 15 to complete the analysis. □

We note here that an $\epsilon$-approximate CDRF mechanism doesn't have to be randomized. In case it is deterministic then the properties above are approximately satisfied in the obvious way.

## 5.2 Identical Machines

While a typical CCC may contain a thousand to a hundred thousand machines, there are typically only a few specific types of machines in the CCC, due to scalability concerns, both for hardware and software. In this setting we can get an $\epsilon$-approximate CDRF mechanism.

For simplicity, we first consider the case when all $m$ machines are identical. In this setting we compute $z = CDRF(d, r_1)$ which is the CDRF expected allocation for a single machine. Since this $z$ is a point in the convex hull of an $n$-dimensional space, we can then find $n$ feasible allocations $z^t \in F_1(d)$ and their associated $\alpha_t$'s such that $z = \sum_{t=1}^{n} \alpha_t z^t$ where each $\alpha_i \geq 0$ and $\sum_{t=1}^{n} \alpha_t = 1$. Then we randomly assign each machine to allocation $z^t$ with probability $\alpha_t$. Define $\epsilon = \sqrt{\frac{12n \log n}{m}}$ and check that all users i, are allocated at least $(1-\epsilon)mz_i$ jobs. If this is not true, then repeat the randomized procedure until it is. We denote this mechanism the Identical Machines-CDRF, or just IM-CDRF.

**Theorem 18.** *Suppose that there are $m$ identical machines. Then IM-CDRF is $O(\sqrt{\frac{n \log n}{m}})$-approximate CDRF. In addition, the number of iterations is less than 2 on average, and more than $\gamma$ with probability less than $2^{-\gamma}$.*

*Proof.* First note that the feasible region of the cluster is the Minkowski sum of the single machine feasible region $m$ times with itself. Since the Minkowski sum of convex hulls is the convex hull of Minkowski sums, if $z = CDRF(d, r_1)$ then $mz = CDRF(d, r)$. Now, let $Y_j$ be random variable for the allocation on the $j$'th machine which takes on value $z^t$ with probability $\alpha_t$.

To simplify the presentation, consider the "normalized variables", $W_{ji} = Y_{ji}/sa_i(d, r_1)$ and $v_i = z_i/sa_i(d, r_1)$ and note that both are contained in the interval $[0, 1]$.

First, we note that $E[W_{ji}] = v_i$ by construction, so $E\left[\sum_{j=1}^{m} W_{ji}\right] = mv_i$. Next we note that $Var[W_{ji}] \leq v_i(1 - v_i)$ since the $W_{ji}$ has mean $v_i$ and is contained on the interval $[0, 1]$ and the maximum variance for such a random variable arises when the random variable takes on only the values 0 and 1. This implies that $Var\left[\sum_{j=1}^{m} W_{ji}\right] \leq mv_i(1 - v_i)$ and thus for the standard deviation $\sigma$ we have $\sigma = \sigma\left[\sum_{j=1}^{m} W_{ji}\right] \leq \sqrt{mv_i(1 - v_i)}$.

Then, using Chernoff's inequality we get that

$$Pr\left[\left|\sum_{j=1}^{m} W_{ji} - mv_i\right| \geq k\sigma\right] \leq 2e^{-k^2/4}.$$

Setting $k^2 = 12\log n$ gives us

$$Pr\left[\left|\sum_{j=1}^{m} W_{ji} - mv_i\right| \geq k\sigma\right] \leq 2/n^3.$$

We then apply a union bound to obtain

$$Pr\left[\left\{\left|\sum_{j=1}^{m} W_{ji} - mv_i\right| \geq k\sigma\right\} \quad \forall i\right] \leq 2n/n^3 = 2/n^2$$

which is less than $1/2$ for $n \geq 2$.

To complete the proof we note that

$$Pr\left[\left|\sum_{j=1}^{m} W_{ji} - mv_i\right| \geq k\sigma\right] = Pr\left[\left|\frac{\sum_{j=1}^{m} W_{ji} - mv_i}{mv_i}\right| \geq \frac{k\sigma}{mv_i}\right]$$

which is the fractional difference between the number of jobs allocated to user $i$ under IM-CDRF and the expected number allocated under CDRF. Thus, we see that the $\epsilon$ in the approximation bound is given by

$$\epsilon \leq \frac{k\sigma}{mv_i} \leq \frac{\sqrt{12\log n\ mv_i(1-v_i)}}{mv_i} \leq \frac{\sqrt{12\ \log n}}{\sqrt{m/n}} = \sqrt{\frac{12\ n\log n}{m}}$$

which is the desired bound, where we used the fact that $1-v_i < 1$ and $v_i > 1/n$ by resource sharing.

Since the probability of this bound being exceeded is less than $1/2$, the number of iterations is bounded by a geometric distribution, leading to the stated bound on iterations. Lastly, ex-ante Pareto optimality is immediate, since all the $z^t$'s are Pareto optimal and lie on the face that contains the CDRF allocation. □

We can extend this to the case where there are several classes of identical machines, but computing the IM-CDRF allocation separately for each class to get the GIM-CDRF allocation.

**Corollary 19.** *Suppose that there are $t < n^b$ machine types and at least $c$ copies of every machine. Then GIM-CDRF is $O\left(\sqrt{\frac{nb\log n}{c}}\right)$-approximate CDRF. In addition, IM-CDRF requires 2 iterations on average and only requires more than $\gamma$ iterations with probability less than $2^{-\gamma}$.*

*Proof.* The constant in the approximation is increased by a factor of $2^{1/2}$, since in order to guarantee that the tail bounds apply to all classes of machines simultaneously we need to choose $k^2 = (12 + b)\log(n)$ in the proof of 18. □

## 5.3 Deterministic mechanisms

One can improve this mechanism by reducing the randomization. For example, instead of choosing the allocation for each machine independently on can directly assign $\lfloor m\alpha_t \rfloor$ machines to allocation $z^t$ and then ignore the remaining machines. This mechanism has the same properties as IM-CDRF but a somewhat different error bound of $\epsilon = O(n^2/m)$, and also is not Pareto Optimal.

Thus, in a common setting, we can directly construct approximate deterministic allocations. In particular, one key property of GIM-CDRF is its simplicity as an algorithm: One only needs to find the CDRF allocation on a single machine of each type and then GIM-CDRF provides a simple algorithm for allocating all the machines. However, for more general sets of machines computing a good allocation is more complex. In the following, we provide a general approximation theorem for arbitrary sets of machines. Our analysis relies on a famous result in convex analysis [14]. This result is non-constructive; it shows the existence of a good approximate CDRF deterministic mechanism, but does not provide a reasonable way of computing it. There is a constructive version of this result ([15]) but it comes with worse approximation bounds and doesn't solve our key algorithmic issue: computing maximum allocations over the feasible region is NP-hard.

Given a set $S_i$ define the inner radius, $IR(S_i)$ to be the smallest value of $\rho$ such that for any point $y \in CH(S_i)$ the ball of radius $\rho$ centered at $y$ will contain a subset of $S_i$ whose convex hull contains $y$. For example, in Figure 4.3 the inner radius of the feasible region is $\sqrt{2}$. The ball with radius $\sqrt{2}$ with center $CDRF(d, r)$ contains $(1, 3)$ and $(2, 2)$, whose convex combination can give $CDRF(d, r)$.

Another example, where we can see that a point on the convex hull can be far from any point in the feasible region is the following: Consider a cluster with $n + 1$ users and $n + 1$ resources. Each user $i$, $i = 1 \ldots n$, demands 1 unit of resource $i$ and 1 unit of resource 0. Player 0 demands $n$ units of resource 0. Assume that this cluster has $n$ machines, where each machine $j$ has $n$ units of resource $j$ and $n$ units of resource 0. For every machine $j$, $(1, 0, \ldots, 0) \in F_j(d)$ and $(0, \ldots, 0, n, 0, \ldots, 0) \in F_j(d)$ where $n$ is in the $j$-th position. $\left(\frac{n}{2}, \frac{n}{2}, \ldots, \frac{n}{2}\right) \in CH(F(d))$ is the CDRF allocation. The closest

integral allocation is $(0, \frac{n}{2}, \ldots, \frac{n}{2})$, which is clearly not Pareto optimal, since $(0, n, n, \ldots, n)$ is feasible; however in computing the inner radius the latter point is used. Thus the inner radius here is $2n\sqrt{n}$.

**Theorem 20** (Shapley-Folkman-Starr)**.** *Let $S_1, ..., S_m$ be a family of $m$ compact subsets of $R^n$, $W = \sum_{i=1}^{m} S_i$. Then for any $x \in CH(W)$ there is $y \in W$ such that $\|x - y\|_2^2$ is bounded by the sum of squares of the $n$ largest $IR(S_i)$.*

In our setting, the Shapley-Folkman-Starr theorem states that one can approximate any allocation in the convex hull of the feasible region, such as the CDRF allocation, with an actual (integral) feasible allocation.

We are now ready to define the **Shapley-Folkman-Starr** mechanism: First we compute the CDRF allocation $z = CDRF(d, r)$. Remember $z_i$ is the expected number of containers user $i$ gets. Next, from the definition of Minkowski sums and the fact that the convex hull of the sum is equal to the sum of the convex hulls, we can decompose $z = \sum_{j=1}^{m} z_j$ where $z_j \in CH(F_j(d))$. Since $z$ is Pareto optimal, all of the $z_j$'s must also be Pareto optimal and furthermore they must lie on a face of $CH(F_j(d))$. This face is composed of at most $n$ points in $F_j(d)$, which we will denote $G_j$; $G_j$ is just the set of (Pareto optimal) allocations on machine $j$ that can span $z_j$. Then, we apply the Shapley-Folkman-Starr theorem to the Minkowski sum of the $G_j$'s to approximate $z$. We call the resulting allocation $y = SFS\,(d, r)$ the SFS mechanism.

In order to prove a reasonable bound on the SFS mechanism we need to make some assumptions about the heterogeneity of the machines. For example, if one machine is much larger than all the other machines then approximations will be difficult. Define $I^*$ to be the maximum inner radius out of all the $F_j(d)$'s: $I^* = \max_j IR(F_j(d))$. The bound will depend on $I^*$. Next, since the bounds need to be multiplicative, if a user is allocated a small number of jobs, then even a small additive approximation will be problematic. Thus we make the simple, and reasonable, assumption that each user can run at least one job per machine, on average. Thus, with $m$ machines each user will have a stand alone bound of $sa_i \geq m$. Under these assumptions we can prove the following:

**Theorem 21.** *Let $I^*$ be the max inner radius of a feasible region of a machine, and suppose that for each user $sa_i \geq m$. Then SFS $O\left(\frac{n^{\frac{3}{2}}}{m} I^*\right)$-approximates CDRF.*

*Proof.* First, we note that by applying the Shapley-Folkman-Starr bound to the $G_j$'s instead of the $F_j(d)$'s directly, we get a Pareto Optimal allocation. Remember that SFS is deterministic, so ex-ante Pareto Optimal is the same as Pareto Optimal. Next, we see from the Shapley-Folkman-Starr bound that $\|SFS(d,r) - CDRF(d,r)\|_2 \leq \sqrt{n}I^*$. This implies that for each user $i$, $|SFS_i(d,r) - CDRF_i(d,r)| \leq \sqrt{n}I^*$, since $\|\cdot\|_2 \geq \|\cdot\|_\infty$. Thus the fractional difference between the SFS allocation and the CDRF allocation is less than $\frac{\sqrt{n}I^*}{(m/n)} = \frac{n^{\frac{3}{2}}I^*}{m}$. This holds because the fair share of user $i$ is at least $\frac{m}{n}$ by assumption. Thus, we see that the SFS mechanism satisfies the stated approximation ratio. □

# Bibliography


[1] Arka A. Bhattacharya, David Culler, Eric Friedman, Ali Ghodsi, Scott Shenker, and Ion Stoica. Hierarchical scheduling for diverse datacenter workloads. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 4:1–4:15, New York, NY, USA, 2013. ACM.

[2] Danny Dolev, Dror G. Feitelson, Joseph Y. Halpern, Raz Kupferman, and Nathan Linial. No justified complaints: On fair sharing of multiple resources. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 68–75, New York, NY, USA, 2012. ACM.

[3] L. E. Dubins and E. H. Spanier. How to cut a cake fairly. *The American Mathematical Monthly*, 68(1):pp. 1–17, 1961.

[4] Eric Friedman, Ali Ghodsi, and Christos-Alexandros Psomas. Strategyproof allocation of discrete jobs on multiple machines. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 529–546. ACM, 2014.

[5] Eric J Friedman, Ali Ghodsi, Scott Shenker, and Ion Stoica. Strategyproofness, leontief economies and the kalai-smorodinsky solution, 2011. Manuscript.

[6] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. Multi-resource fair queueing for packet processing. *SIGCOMM Comput. Commun. Rev.*, 42(4):1–12, August 2012.

[7] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on*

*Networked Systems Design and Implementation*, NSDI'11, pages 24–24, Berkeley, CA, USA, 2011. USENIX Association.

[8] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[9] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *INFOCOM*, pages 1206–1214, 2012.

[10] Ehud Kalai and Meir Smorodinsky. Other solutions to nash's bargaining problem. *Econometrica*, 43(3):pp. 513–518, 1975.

[11] Jin Li and Jingyi Xue. Egalitarian division under leontief preferences. *Economic Theory*, 54(3):597–622, 2013.

[12] linuxcontainers.org. http://linuxcontainers.org/, 2014.

[13] David C. Parkes, Ariel D. Procaccia, and Nisarg Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pages 808–825, New York, NY, USA, 2012. ACM.

[14] Ross M. Starr. Quasi-equilibria in markets with non-convex preferences. *Econometrica*, 37(1):pp. 25–38, 1969.

[15] Ross M Starr. Approximation of points of the convex hull of a sum of sets by points of the sum: an elementary approach. *Journal of Economic Theory*, 25(2):314–317, 1981.

[16] Wei Wang, Baochun Li, and Ben Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. *CoRR*, abs/1308.0083, 2013.