Computational Game Theory **Spring Semester, 2011/2012**

Dominant Resource Fairness: Fair Allocation of Multiple Resource Types[1]

Final Project Report By: Yael Amsterdamer ID: 037723277

In this report I will first describe the work of Ghodsi et. al and their contributions. I will explain the notion of Dominant Resource Fairness in policies for the allocation of multi-type resources, and describe the analysis of its valuable properties. In the second part I will discuss my attempts in extending these results in 3 new directions: analysis of DRF in the discrete case (when only whole tasks can be executed), resource utilization of DRF solutions, and DRF with assignment constraints.

1 Paper Overview

1.1 Resource Allocation - Introduction

The problem of resource allocation is a classical one: consider a situation in which a few players and common resources are involved. The players provide their demands for the resources, and a decision must be made accordingly, on how to split the resources. Naturally, problems arise only when there are not enough resources to satisfy the demands of all the players. In such situations, there may be many possible options on how to split the resources. To illustrate that, consider the following example.

Example 1 *Alice and Bob are using a shared server for their computational tasks, which has 18GB of RAM. Alice has MATLAB scripts she wants to execute, each requires approximately 4GB of memory during its execution. Bob wants to convert video files to a different format, and each conversion requires approximately 2GB of memory.*

One option for splitting the resources is according to user demands. *We can give each user the part of resource, memory in this case, proportionally to their demands. This means that Alice gets 12GB of memory, and Bob gets 6GB. This approach is* fair *in the sense that the players get resources according to their needs, and for instance, can execute the same amount of tasks (three). It is unfair in the sense that Alice gets a larger absolute amount of resources. Another downside of this approach is the fact that the players have* an incentive to lie *– the more resources each player demands, the more they get. Therefore, this approach is unstable for cases when players can lie and do not cooperate with each other. In such cases each player will demand as many resources as possible.*

Another option is ignoring completely the player demands, and splitting the resources evenly. It is known as the max-min *approach, as it can be viewed as an attempt to maximize the amount of resources assigned to the player with least resources, then the amount of resources assigned to the second worst player, and so on. In our example this means that both Alice and Bob get 9GB of memory. This approach is fair in the sense that each player gets exactly the same amount of shared resources. Players cannot profit from lying, as their demands are ignored completely. However, the downside is that players with high demands*

[1] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Usenix NSDI*, March 2011.

can execute less tasks. Moreover, this is a very degenerate form of sharing – the players get no benefit from the fact that the resource is shared [2].

The paper of Ghodsi et. al addressed a less-explored aspect of this problem, namely resource allocation in the presence of multiple types of shared resources. In this scenario, which often occurs in reality, players have different demands for each of the resources. For instance, we can extend Example 1 such that each task of Alice and Bob also requires a certain amount of CPU, a certain amount of network bandwidth and so on. It turns out that although this setting is more complicated, it allows the players to get more benefit from the fact that resources are shared. The crux here is that when different players require more of different resources, they can split them such that every player gets more of their most needed resource, and give up a less needed resource in return.

Example 2 *Consider a situation identical to that of Example 1, but now Alice and Bob have additional CPU demands: each task of Alice requires one CPU, and each task of Bob requires 3. The shared server has 9 CPUS. In this case, it is clear that splitting the CPU and memory evenly between Alice and Bob makes no sense; both players would benefit if Alice gets more memory, and Bob gets more CPUs. For instance, if Alice gets* $4/9$ *of the memory, i.e., 8 GB, she only needs 2 CPUs, which are* $1/3$ *of the total amount of CPUs.*

1.2 Properties of a good allocation

Ghodsi et. al refer in their paper to different measures for a good allocation. Some of them are general measurements of solutions in game theory, others are unique to the current settings. We give a brief overview of these properties.

Incentive Compatibility participants cannot improve their situation (e.g., get resources for more tasks) by lying about their demands.

Pareto Efficiency Some participants would lose from any change of allocation.

Envy-freeness No participants will prefer the allocation of another participant over their allocation. In particular, this entails that participants with equal needs will get equivalent allocations.

Sharing Incentive No participants will prefer the allocation of $1/N$ of each resource (for N participants). This means that if each user contributes an equal amount of each resource type, all the users have an incentive to share and not simply use their own resources. This provides us with a minimum allocation guarantee for each user.

Single-resource Fairness The allocation for a single resource type should be equivalent to that of min-max.

Bottleneck Fairness when one resource is demanded most by all the participants, allocation will be equivalent to min-max on this resource, while adjusting the rest of the allocations accordingly.

[2] In the continuous case, where it is possible to execute parts of tasks. In the discrete case they may benefit a little, since the leftover resources after splitting evenly, may be enough for the execution of a few more tasks.

Monotonicity Removing a user or adding a resource can only increase the allocation of the remaining users.

For example, it is a simple exercise to show that min-max, in particular, has all of the properties above.

The first 4 properties are defined by the authors as crucial for a good allocation policy. The last 3 are considered nice-to-have.

1.3 Dominant Resource Fairness

We start by formalizing the problem settings of a multi-type resource assignment, that were explained before intuitively. Assume that we would like to assign resources from n different types to the participants. Define the available resources using the vector $(r_1^{\max}, \ldots, r_n^{\max})$, where $r_j^{\max}$ denotes the capacity of the j-th resource type. Now, for each participant i we define a *demand vector* $(r_1^i, \ldots, r_n^i)$, where r_j^i denotes the amount of resource of type j required by a single task of the i-th user. Note that for now we assume that each participant has only one such demand vector, and that the amount of executed tasks per user is continuous; namely, according to the resource assignment, a user may be able to execute 1.02 tasks, 0.005 tasks, etc.

Definition 3 (Dominant Share and resource) *Assume that the i-th participant is assigned the resources for the execution of exactly a_i tasks. Then the* dominant share *of user i is defined as*

$\max_{j=1}^{n} \frac{r_j^i}{r_j^{\max}}$

The dominant resource *is defined accordingly as*

$\arg\max_{j=1}^{n} \frac{r_j^i}{r_j^{\max}}$

Intuitively, the dominant share is the share of user i of some resource which is the largest relatively to the resource capacity. Note that it is defined in terms of assignment, and not in terms of demand; this definition will be useful later on for the assignment process. The dominant resource definition does not change upon the removal of a_i, which is just a constant. Thus, the dominant resource indicates the type of resource for which the relative demand of user i is the highest.

We now get to the principle of the suggested assignment policy, based on the notions defined above:

Definition 4 (Dominant resource fairness (DRF)) *An assignment of shared resources is* dominant resource fair *if the smallest dominant share is maximized, then the second smallest dominant share is maximized, etc.*

Note that the computation is more complicated than in the classical max-min: when we allocate the dominant resource for some task, we also allocate proportionally other resources.

Example 5 *Recall the settings of Example 2. According to DRF, the optimal allocation is of* 18/7 *tasks for Alice (10.29GB, 2.57CPUs) and of* 8/7 *tasks for Bob (2.29GB, 3.43CPUs). In this assignment, the dominant share of each participant is exactly* 4/7*, and*

since all the CPUs are used, there is no possibility of increasing the share of one participant without reducing the share of the other.

In the paper, Ghodsi et. al prove that DRF has most of the desired properties of a good allocation, mentioned in Sec. 1.2. In particular, it has the first four properties defined as important. The only property not fulfilled by DRF is resource monotonicity. However, they prove that policies for which Pareto efficiency and sharing incentive hold, resource monotonicity cannot hold.

1.4 Achieving DRF

The problem of finding the DRF allocation of resources can now be formalized using the following linear program. Let x_i be a variable that determines the number of tasks assigned to the i-th participant. Let n be the number of resource types and N the number of participants.

Maximize x_1

Subject to

$$\begin{cases} \sum_{i=1}^{N} r_k^i \cdot x_i \leq r_k^{\max} & (k = 1, \ldots, n) \\ \max_{k=1}^{n} \frac{r_k^i}{r_k^{\max}} \cdot x_i = \max_{k=1}^{n} \frac{r_k^{i+1}}{r_k^{\max}} \cdot x_{i+1} & (i = 1, \ldots, N-1) \end{cases}$$

The first constraint enforces that the total assigned resources do not exceed the available resources; the second constraint enforces that dominant share is equal for every participant, for fairness purposes. Under those constraints, we attempt to maximize the number of tasks assigned to the first user, which is equivalent to maximizing the dominant share of this user, and, due to the second constraint, to maximizing all the dominant shares. Note that the program above encodes no upper limit on the x_i values, which may exist in practice (e.g., if Alice has only a fixed amount of tasks to execute).

One obvious option for finding the optimal DRF allocation is by solving the linear program presented above. However, in order to investigate the properties of such allocation, the authors of the paper discuss additional options.

The first option, which appears in the paper, is a simple, iterative algorithm, for the allocation of tasks based on the DRF notion:

- Pick the user with the current minimal dominant share
- If there are enough resources for his next task
  - Launch the user task
  - Update the current resource allocation, resources per user, and user dominant shares

However, it seems that this algorithm is slightly over-simplified. For instance, if there are not enough resources for the user with minimal dominant share, the process is stuck. Instead, by the principle of DRF, the algorithm should have then turned to improve the allocation of the user with the second-smallest dominant share, and so on. Moreover, this algorithm only allocates complete tasks, and is not relevant for the continuous scenario

which is the focus of the paper. Finally, as mentioned in the paper, the algorithm also ignores events of task finishing, when resources are freed and should be re-allocated.

Here we will focus on the more precise algorithm for continuous allocation, provided in the full version of the paper. This is a *progressive filling* algorithm, adapted from the area of networking to resource sharing and to DRF. All of the proof for the DRF properties are based on the correctness of this algorithm. This algorithm is simple as well, and is based on the assumption that arbitrarily small fractions of tasks/resources may be allocated.

- Initialize U to contain all the users
- While U is not empty
  - While no resource is saturated
    - Gradually and equally increase the dominant share of all the users in U.
    - Allocate proportionally the other, non-dominant resources for every user.
  - Remove from U all users with positive demands for saturated resources

This algorithm achieves DRF by increasing as much as possible the dominant share of all the users.

1.5 Competing Policies

Recall that DRF fulfills all of the important properties of a good resource sharing policy. We will now compare DRF with two other policies, and show that not all of the important properties hold for them.

1.5.1 Asset Fairness

The goal of an *asset fair* policy is to balance the *total* fraction of resources allocated for each user. The policy can be described by the following linear program.
Maximize x_1
Subject to

$$\begin{cases} \sum_{i=1}^{N} r_k^i \cdot x_i \leq r_k^{\max} & (k = 1, \ldots, n) \\ \sum_{k=1}^{n} \frac{r_k^i}{r_k^{\max}} \cdot x_i = \sum_{k=1}^{n} \frac{r_k^{i+1}}{r_k^{\max}} \cdot x_{i+1} & (i = 1, \ldots, N-1) \end{cases}$$

The variables and constants are defined in the same manner as the linear program of DRF. The maximization target and the first constraint are also the same, but in the second constraint, *sum* is used instead of max. This way we enforce that the total fraction of resources in the same for every user, rather than the dominant share.

The following example shows that Asset Fairness does not fulfill the sharing incentive property.

Example 6 *Assume that a server has 10GB of memory and 10 CPUs. Alice's demand vector is* $(3GB, 1\ CPUs)$*, and Bob's is* $(2GB, 2\ CPUs)$*. An asset-fair allocation for this*

Property	Allocation Policy		
	Asset	CEEI	DRF
Incentive Compatibility	✓		✓
Pareto Efficiency	✓	✓	✓
Envy-freeness	✓	✓	✓
Sharing Incentive		✓	✓
Single-resource Fairness	✓	✓	✓
Bottleneck Fairness		✓	✓
Population Monotonicity	✓		✓
Resource Monotonicity			

Table 1: Summary of the properties

situation will assign 6GB and 2 CPUs for Alice, and 4GB and 4 CPUs for Bob. This way, both of them get a total fraction of $8/10$ *of resources. However, if Bob had half of the resources, he could have used 5GB and 5 CPUs, to execute an additional 0.5 of a task.*

1.5.2 Competitive Equilibrium from Equal Incomes (CEEI)

Another notion of fairness, taken from microeconomics, is the Competitive Equilibrium from Equal Incomes (CEEI, for short). In this policy, each user gets $1/N$ of the resources, and trades them in a perfect market. The obtained solution is a *Nash bargaining solution*, which maximizes the product of some utility function. In our case, the utility of each user can the number of tasks executed by him, or, e.g., his dominant share. CEEI can be described by the following program (nonlinear, this time)

Maximize $\prod_{i=1}^{N} x_i$

Subject to

$$\sum_{i=1}^{N} r_k^i \cdot x_i \leq r_k^{\max} \quad (k = 1, \ldots, n)$$

However, Ghodsi et. al prove in the paper, using a counter-example, that CEEI is not incentive compatible. They show that in some cases, users that ask for resources that they do not need, may actually increase their allocation.

1.5.3 Summary of the properties

In the full version of the paper, the authors continue and prove that some of the nice-to-have properties do not hold for asset fairness and CEEI. Table 1 summarizes the results from the paper. Note that the monotonicity property is split to population monotonicity (what happens when a user is removed) and resource monotonicity (what happens when a resource is added). In particular, DRF is the only policy for which all four important properties, as well as three of the nice-to-have properties, hold. No policy fulfills resource monotonicity.

2 New Contributions

I next describe the ideas and directions I explored that arise from the paper of Ghodsi et. al. I will discuss the challenges that arose on the different directions, and some new results I have obtained.

2.1 DRF in the discrete case

In the overview of the work by Ghodsi et. al above, we have learned about the important properties of DRF, given as an indication that DRF is indeed a good and useful allocation policy. However, all of the results mentioned above were proven in the paper only for the continuous case, i.e., when any fraction of a task may be executed. The discrete case, when only whole tasks may be executed is also mentioned. For this case, only an upper bound on the difference in the allocation with respect to the continuous case is given.

However, this raises a question that is not discussed in the paper (and not in the full version): *Do the properties still hold in the discrete scenario?* It turns out, that the answer is no. The next propositions show which properties hold and which do not. For that, assume that we can obtain DRF in a discrete solution.

Proposition 7 *DRF in the discrete case is not envy-free.*

Proof: Consider the case when only one type of resource is available, with a capacity of x. Let there be two users, where the first has the demand vector $(3x/4)$ and the second has $(x/4)$. It is easy to show that the dominant-resource-fair discrete solution gives to the first and second users $3x/4$ and $x/4$ of the resource respectively. If we do not give the first user $3x/4$ of the resource, he will not be able to execute any tasks, and the minimal dominant share would be worse – 0. Considering the DRF allocation, the second user would prefer the allocation of the first user, which would allow him to improve his situation and execute two more tasks. Thus, DRF in the discrete case is not envy-free. □

Proposition 8 *DRF in the discrete case is Pareto efficient.*

Proof: Assume that in some discrete dominant-resource-fair solution at least one user could allocate at least one more task, without reducing the task allocation of other users. Among such users, take the one u with the smallest dominant share. Assume that u has the n-th smallest dominant share. Then allocating one more task for u would improve his allocation, which contradicts the fact that our previous allocation was dominant-resource-fair. □

Proposition 9 *Sharing incentive does not hold for DRF in the discrete case.*

Proof: Consider again the example from the proof of Prop. 7. In the dominant-resource-fair discrete solution the second user gets to execute one task, whereas if he had half of the resource, he could have executed two tasks. Thus, this user would prefer to get 1/2 of the resources, and has no incentive to share. □

Proposition 10 *DRF in the discrete case is not incentive compatible.*

Property	Allocation Policy	
	Cont. DRF	Discrete DRF
Incentive Compatibility	✓	✗
Pareto Efficiency	✓	✓
Envy-freeness	✓	✗
Sharing Incentive	✓	✗

Table 2: Summary of the discrete case properties

Proof: Let there be a single resource with capacity x, and two users with the demand vector $(x/4)$. The discrete DRF solution gives each user half of the resource, and each user would be able to execute 2 tasks. However, if the first user lies that his demand vector is $(3x/4)$, he will get $3x/4$ of the resource and will be able to execute 3 tasks (as we saw in the proof of Prop 7). Thus, this user has an incentive to lie. □

The results are summarized in Table 2.

All of the proofs above, except from the positive result w.r.t. Pareto efficiency, actually use a single resource. This means, in particular, that min-max in the discrete case does not fulfill all of the desired properties. Perhaps, the conclusion is not that DRF is not a good policy for the discrete case, but rather that we need to redefine or relax the desired properties, so that they will reflect what is reasonable to expect from a multiple-resource sharing policy in the discrete case.

2.2 Bounding the efficiency of DRF

One possible topic that was suggested during my presentation in class, was finding a bound on how many resources are left unutilized as a result of DRF. Indeed, as mentioned in Sec. 1.4, the simplified algorithm for discrete allocation that appears in the paper misses some important points: it may get stuck when there are not enough resources for the user with the smallest dominant share. Consequently, some resources may be left that could have been utilized by other users. However, this only happens because the algorithm does not really achieve DRF. It maximizes the minimal dominant share, but not the second smallest dominant share, etc.

Consider the allocation resulting from progressive filling (also described in Sec. 1.4). Here, it is clear that even if some resources are left unutilized, no user could have benefitted from them without changing the existing allocations. Otherwise, this implies that the user has no demands for saturated resources. This means that the progressive filling algorithm could not have completed, and leads us to a contradiction. Using the same arguments, I can show that DRF is Pareto efficient - no users can improve their situation without some other user losing.

However, other policies such as CEEI may induce better resource utilization. In the case of CEEI, we saw that this comes at the expense of incentive compatibility. We can ask: How worse can the resource utilization of DRF be vs. the optimal resource utilization? The answer is provided in the following proposition.

Proposition 11 *The resource utilization of DRF can be unboundedly small with respect to the optimal resource utilization.*

Proof: Let there be x resource types, and y users. For simplicity, assume that the capacity of all the resources is given and demanded in percentages; i.e., the vector of capacities

is $(100\%, \ldots, 100\%)$. Let the demand vector of user no. 1 be $(100/y\%, \ldots, 100/y\%)$, and the demand vectors of users 2 to y be $(100/y\%, 0, 0, \ldots, 0)$. In other words, the first user requires the same percentage of every resource, and the other users require only their share of the first resource. Note that in this case, the first resource is a dominant resource for all the users. By the bottleneck fairness property of DRF, the dominant-resource-fair allocation will give an equal share to all the users from the most demanded resource, i.e., every user will get $100/y\%$ of the first resource (and the rest of the resources proportionally), which will allow every user to execute exactly one task.

In terms of resource utilization, the amount of resources utilized by the allocation described above is:

$$\underbrace{1 \cdot y \cdot 100/y}_{\substack{\text{Utilization of the} \\ \text{first resource}}} + \underbrace{(x-1) \cdot [1 \cdot 100/y + (y-1) \cdot 0]}_{\text{Utilization of the rest of the resources}} = 100 + \tfrac{100(x-1)}{y}$$

In contrast, in the allocation that is optimal w.r.t. resource utilization, all the resources $(100x\%)$ are given to the first user, for the execution of exactly y tasks. We get:

$$\frac{100 + \frac{100(x-1)}{y}}{100x} = \frac{y + x - 1}{xy}$$

The limit of this expression when both x and y go to infinity is 0. This means that the fraction of resources utilized by a resource-dominant-fair solution can be arbitrarily small w.r.t. the optimal resource allocation. □

2.3 Adding placement constraints

One open problem from the paper that I have considered investigating was that of placement constraints: assume that in addition to the resource demands, each task also has placement constraints. Namely, every task can run only on particular machines and not on other machines.

In the simplest case, there are distinct sub-groups of machines, and each task can be executed on exactly one of the groups. In this case, no new analysis is required: If there are n resource types and g groups, the size of the capacities vector and demand vectors would be $n \times g$, where each entry refers to a particular resource type on a particular group. Given these capacities and demands, we can find the DRF solution as we did before.

In the more complicated case, the sub-groups of machines may be overlapping. We can describe the new situation using the following linear program.

Maximize x_1

Subject to

$$\begin{cases} \sum_{i=1}^{N} \sum_{m \in M(i)} r_k^i \cdot x_{i,m} \leq r_{k,m}^{\max} & (k = 1, \ldots, n) \\ \sum_{m \in M(i)} x_{i,m} = x_i & (i = 1 \ldots N) \\ \max_{k=1}^{n} \frac{r_k^i}{r_k^{\max}} \cdot x_i = \max_{k=1}^{n} \frac{r_k^{i+1}}{r_k^{\max}} \cdot x_{i+1} & (i = 1, \ldots, N-1) \end{cases}$$

As before, the variable x_i represents the number of tasks assigned to user no. i. $M(i)$ is a constant representing the set of machines on which the tasks of i can be executed. r_k^i, as before, is a constant representing the demand of each task of user no. i from resource no. k. $x_{i,m}$ is a variable representing the number of tasks of user i executed on the machine m. The second constraint of the program enforces that x_i is the sum of tasks of i executed on any machine available for i. $r_{k,m}^{\max}$ is the capacity of resource k on machine m. The first constraint enforces that the capacity of every machine is not exceeded by the executed tasks. Finally, the third equation is unchanged from the original DRF program, and enforces that the dominant share is equal for all of the users. Solving this linear program will give a solution to DRF on multiple machines.