Fair Allocation of Heterogeneous and Interchangeable Resources

Xiao Sun[1], Tan N. Le[1], Mosharaf Chowdhury[2], and Zhenhua Liu[1]

[1]Stony Brook University, NY, E-mail:{xiao.sun, tan.le, zhenhua.liu}@stonybrook.edu

[2]University of Michigan, MI, E-mail: mosharaf@umich.edu

Abstract

Motivated by the proliferation of heterogeneous processors such as multi-core CPUs, GPUs, TPUs, and other accelerators for machine learning, we formulate a novel multi-interchangeable resource allocation (MIRA) problem where some resources are interchangeable. The challenge is how to allocate interchangeable resources to users in a sharing system while maintaining desirable properties such as sharing incentive, Pareto efficiency, and envy-freeness. In this paper, we first show that existing algorithms, including the Dominant Resource Fairness used in production systems, fail to provide these properties for interchangeable resources. Then we characterize the tradeoff between performance and strategyproofness, and design the Budget-based (BUD) algorithm, which preserves Pareto efficiency, sharing incentive and envy-freeness while providing better performance over currently used algorithms.

1. INTRODUCTION

Cloud computing jobs need multiple types of resources. For instance, tasks in a MapReduce job require a minimum amount of CPU, memory, disk, and network bandwidth to make progress. Unlike traditional single-resource allocation algorithms that focus on only CPUs or network bandwidth, a multi-resource allocation algorithm *simultaneously* allocates more than one resource among jobs from different users. Formally, a demand vector $\overrightarrow{d_k} = \langle d_k^1, d_k^2, \ldots, d_k^n \rangle$ denotes user-k's demands on n resources, where $d_k^i \leq 1$ represents her demand on the i-th resource normalized by the resource capacity. We further scale $\overrightarrow{d_k}$ to make her *dominant* resource $\max_j d_k^j = 1$. The *progress* α_k of user-k is defined as $\alpha_k = \min_i \left\{ \frac{a_k^i}{d_k^i} \right\}$ (Leontief production function), where $a_k^i \leq 1$ represents user-k's allocated fraction on resource-i.

Among existing multi-resource allocation algorithms, the Dominant Resource Fairness (DRF) [2] and its variants [1] are probably the most widely-used, e.g., in Apache YARN and Spark. DRF extends the max-min fairness from single resource to the multi-resource allocation. Specifically, it allocates resources proportionally to their demand vectors, i.e., $\overrightarrow{a_k} = \alpha_k \cdot \overrightarrow{d_k}$ and equalizes α_k among all users. DRF guarantees desirable properties such as sharing incentive, strategyproofness, envy-freeness, Pareto efficiency, among others.

Consider a simple system with CPUs and memory as resources. There are two users with demands $\overrightarrow{d_1} = \langle 1, \frac{1}{8} \rangle$ and $\overrightarrow{d_2} = \langle \frac{1}{2}, 1 \rangle$. This means user-1's dominant resource is CPU, and for each fraction of CPU, 1/8 fraction of memory is needed to make full progress. Similarly, memory is the dominant resource for user-2, where 1/2 fraction of CPU is needed for each fraction of memory allocated. DRF allocates $\overrightarrow{a_1} = \langle \frac{2}{3}, \frac{1}{12} \rangle$ and $\overrightarrow{a_2} = \langle \frac{1}{3}, \frac{2}{3} \rangle$ and $\alpha_1 = \alpha_2 = \frac{2}{3}$. Note that this progress is significantly higher than the $\frac{1}{2}$ under equal sharing of the resources.

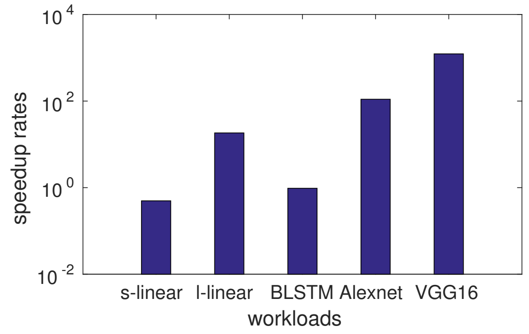

Figure 1: Jobs have distinct efficiencies in utilizing GPU. The speedup rates are ratios between job completion time using one CPU core and that using one GPU.

Other allocation algorithms such as Competitive Equilibrium from Equal Incomes (CEEI) and asset fairness fail to provide some key properties in multi-resource allocation. In particular, CEEI requires market clearance, which cannot be achieved under Leontief production function.

There is another dimension of complexity being overlooked: multiple devices can actually be used in an interchangeable way to fulfill the same resource demand. In particular, both CPUs and GPUs can be used for the computation. Generally speaking, CPU is more efficient for complicated computations but it has a limited number (10s) of threads. In contrast, GPU is used primarily for highly parallelized (thousands of cores per GPU), simple computations. GPUs are becoming popular for machine/deep learning, massive data mining, etc. Google even developed a specialized FPGA-based processor TPU for its AlphaGo AI system.

The key challenge is that different types of jobs may have distinct speedup rates from GPUs. Figure 1 illustrates our results from real experiments using both simple workloads such as linear regression and standard machine learning benchmarks. While GPU is very efficient in accelerating applications such as VGG16, AlexNet and l-linear, it is not suitable for other applications such as s-linear and BLSTM.

Our contributions are three-fold. Motivated by observations on real systems, we first formulate a new Multi-Interchangeable Resource Allocation (MIRA) problem. Then we uncover the hard tradeoff between desirable properties in MIRA and inefficiencies of existing solutions. Finally, we design the Budget-based Algorithm that preserves desirable properties and outperforms existing solutions.

2. MODELING

The interchangeable resource allocation is denoted by the quadruple (N, Q, E, D). $N = \{1, 2, \cdots, n\}$ denotes the set of users. $Q = \{r_1^1, \ldots, r_{M_1}^1, r_1^2, \ldots, r_{M_2}^2, \ldots, r_1^q, \ldots, r_{M_q}^q\}$ is the set of resources, which is categorized into q interchangeable pools, e.g., computation, storage, networking. For the j-th pool, there are M_j types of resources within the pool that are interchangeable. The system has $M = \sum_{i=1}^{q} M_i$ resources in total. $E = \{e_1, e_2, \ldots, e_n\}$, where each e_i is an $M \times M$ matrix with elements $e_i(x, y)$ representing the relative efficiency of resource-x compared to resource-y if they are interchangeable and is set to be zero otherwise. Finally, $D = \{\overrightarrow{d_1}, \overrightarrow{d_2}, \ldots, \overrightarrow{d_n}\}$ represents the set of demands. $\overrightarrow{d_i}$ is a q-vector where d_{ij} representing user i's demand on the j-th resource pool, e.g., computation.

Consider a simple example with $M = 3$ resources: CPU, GPU and memory. The system has 6 CPUs, 4 GPUs, and 40 GB memory. There are $q = 2$ pools, one of computation and one of storage, where CPUs and GPUs are in the first pool and they are interchangeable. Two users arrive at the system with demand vectors on computation and storage: $\overrightarrow{d_1} = \langle 1/4, 1\rangle$ means that User 1 needs 1/4 unit of computation (measured in CPU) for each unit of memory; $\overrightarrow{d_2} = \langle 1, 1/16\rangle$ means that User 2 needs 1/16 unit of storage for each unit of computation. Two matrices $e_1 = \begin{pmatrix} 1 & 2 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $e_2 = \begin{pmatrix} 1 & 1/80 & 0 \\ 80 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ represent that resource 1 (CPU) and resource 2 (GPU) are interchangeable and they are not interchangeable with resource 3 (memory). To process User 1's jobs, CPU is twice faster than GPU, while for User 2, GPU is 80× faster than CPU.

An allocation is a mapping from (N, Q, E, D) to an allocation $A = \{\overrightarrow{a_i}, \forall i \in N\}$, where $\overrightarrow{a_i} = \langle a_{i1}^1, \ldots, a_{iM_1}^1, \ldots, a_{i1}^q, \ldots, a_{iM_q}^q\rangle$ denotes the allocated resources to user i. Given this allocation, user i's total resource in pool j as measured by the first resource in that pool is

$$a_{ij} = \sum_{k=1}^{M_j} a_{ik}^j e_i \left(k, \sum_{l=1}^{j-1} M_l + 1\right).$$

The progress of user i is again defined as

$$\alpha_i = \min\left\{\frac{a_{i1}}{d_{i1}}, \ldots, \frac{a_{iq}}{d_{iq}}\right\}.$$

With previous example, consider the allocation $\overrightarrow{a_1} = (5, 0, 20)$ and $\overrightarrow{a_2} = (0, 4, 20)$ which leads to progresses of $\alpha_i = 20$ and $\alpha_j = 320$. It is envy free as neither user prefer the other's allocation. It also provides sharing incentive because both users are better than equally sharing the resources, which results in progresses $\alpha_i = 16$ and $\alpha_j = 163$.

Now we characterize optimal solutions based on the following properties considered in algorithmic game theory and multi-resource allocation.

Pareto Efficiency (PE): No user can increase her progress without hurting the progress of at least another user.

Sharing Incentive (SI): Each user is no worse off by sharing than exclusively using $\frac{1}{n}$ of the system all the time.

Envy-Freeness (EF): No user prefers the allocation of another user for higher progress.

Strategy-proofness (SP): No user is able to benefit by lying about her resource demands.

SI provides some performance isolation because it guarantees a minimum utilization for each user irrespective of the demands of other users. Without SI, users may leave the system with their own shares, leading to the collapse of the sharing system. EF embodies the notion of fairness.

If PE is not met, there exists a possible (Pareto) improvement on the system's resource utilization. Without SP, users can gain in the allocations by lying, which may lead to inefficiency in the allocations and hurt other properties.

While DRF satisfies these properties for non-interchangeable allocation, it fails to provide most of them under interchangeable environment. Formally, we have

LEMMA 2.1. *There exists cases where DRF fails to provide PE, SI or SP, even with only one resource pool consisted of two interchangeable resources.*

Consider the simple case with 3 resources CPU, GPU and memory, where CPU and GPU are interchangeable. Suppose the system has n users, where the first $n-1$ users prefer GPU and the last user prefers CPU. Intuitively, the first $n-1$ users would submit job profile with GPU and memory. However, based on the DRF scheme, if the bottleneck is computation, they each can get $\frac{1}{n-1}$ of GPU, depending on its efficiency matrix, the computation resource they get could be worse than equal sharing. This violates SI. Meanwhile, the CPU utilization is at most $\frac{1}{n-1}$, so it turns out neither computation resources or memory is saturated, which violates PE. Finally, some user may lie about her demand to request both resources and could potentially get more computation resource if there is free space in CPU. Therefore the allocation is not strategyproof.

Surprisingly, there is a hard tradeoff among these basic properties. In particular, we can show for any allocation, if it provides sharing incentive and Pareto efficiency, it cannot be strategyproof. Conversely, if it is strategyproof, sharing incentive and Pareto efficiency cannot be provided simultaneously. There is one exception: if all users have homogeneous relative efficiencies across all resources, then the problem degenerates to a traditional multi-resource allocation.

LEMMA 2.2. *No allocation can satisfy (i) PE and SI, and (ii) SP simultaneously unless there exist non-zero scalars $k_1, k_2, \ldots, k_n$, so that the relative efficiencies $k_1e_2 = k_2e_2 = \cdots = k_ne_n$ for all n users.*

Now we briefly discuss the intuition behind this lemma. Consider the environment with CPU, GPU and memory defined above. Two users i and j have the true relative efficiency of GPU compared to CPU g_i and g_j, we assume $g_i < g_j$. Let the reported relative efficiency be $\tilde{g}$. PE implies that user i should utilize CPU first while user j utilizes GPU first. SI requires that both users get at least $\frac{1}{2}(1 + g_i)$ and $\frac{1}{2}(1 + g_j)$ computation, respectively. As a consequence, user i can report a $\tilde{g}_i$ that $\tilde{g}_i = \tilde{g}_g - \delta_1$ for small $\delta_1 > 0$ to get more resources, which is ensured by SI. User j can also manipulate its demand to counter i's movement by lowering its $\tilde{g}_j$ to $\tilde{g}_j = \tilde{g}_i + \delta_2$ for small $\delta_2 > 0$. As a consequence, there does not exists a point $(\bar{g}_i, \bar{g}_j)$ where both users are satisfied.

3. BUDGET-BASED ALGORITHM

BUD consists of two components. The system operator determines a per unit tag price for each resource using Algorithm 1. The price is the same for each user, i.e., no price discrimination. Denote resource prices by $P = \{p_1^1, \ldots, p_{M_1}^1, p_1^2, \ldots, p_{M_2}^2, \ldots, p_1^q, \ldots, p_{M_q}^q\}$.

Each user has the same virtual budget b. The payment for each pool j is $\sum_{k=1}^{M_j} a_{ik}^j p_k^j$. For the case of two interchangeable resources (CPU, GPU) and another non-interchangeable resource (memory), the budget is the maximum money that can be spent on either CPU and GPU combined, or memory. Each user-i solves the following linear programming to maximize her progress, where $a_{ij} := \sum_{k=1}^{M_j} a_{ik}^j e_i(k, \sum_{l=1}^{j-1} M_l + 1)$ denotes aggregated resources evaluated by the first resource in each pool.

$$\begin{array}{lll} \max & \alpha_i & \\ \text{subject to} & \sum_{k=1}^{M_j} a_{ik}^j p_k^j \le b \ \forall j & \\ & \alpha_i \le \frac{a_{ij}}{d_{ij}} \quad \forall j & \\ & a_{ik}^j \ge 0 \quad \forall j,k & \end{array} \tag{1}$$

For the case of two interchangeable resources (CPU, GPU) and another non-interchangeable resource (memory), let the load of 3 resources be (H_c, H_g, H_m). The algorithm to decide pricing is:

Algorithm 1 Price calculation

```
 1: Sort users based on their $\beta_i := e_i(2,1)$ in ascending order.
 2: Initialization: $b := 1$, $p_c := 1$, $p_g := \beta_n$, $p_m = 1 + \beta_n$
 3: Each user solves LP (1) and update $H_c, H_g, H_m$ and $I_g$ (smallest index of a user using GPU)
 4: while $H_g \neq H_c$ do
 5:     if $H_g < H_c$ then
 6:         $p_g := \beta_{I_g}$, $p_m = p_g + 1$ (move user $I_g - 1$ to GPU)
 7:         Update $H_c, H_g, H_m, I_g$ using (1)
 8:     else
 9:         Adjust user $I_g$'s allocation from GPU to CPU or
10:         Decrease $p_g$ while keeping $p_m = p_g + 1$ until $H_g = H_c$
11:     end if
12: end while
13: $b = \frac{b}{\max(H_c, H_g, H_m)}$.
```

Lemma 3.1. *BUD is PE, SI and EF for two interchangeable resources.*

The proof idea is briefly discussed here. When BUD finds the point where the load of CPU and GPU equalize, the price (p_c, p_g, p_m) is determined. By scaling the budget, either computation or memory becomes saturated, which provides Pareto Efficiency. Regarding sharing incentive, we show that any user i can afford $\frac{1}{n}(1+\beta_i)$ computation and $\frac{1}{n}$ memory. If memory is saturated, then there exists at least one user whose memory share is no less than $\frac{1}{n}$, given the unit price for memory, every user should have at least $\frac{1}{n}p_m$ budget. For this amount of budget, users can also buy $\frac{1}{n}$ of CPU and GPU, no worse than equal sharing. Similar argument can be applied when computation is the bottleneck. Envy-freeness is a natural consequence due to the same budget for each user and indiscriminate tag prices, which ensures that every user can afford the allocation of others.

Interestingly, the traditional DRF allocation is a special case for our BUD algorithm. When there are no interchangeable resources, every resource has a unit price. Consider the system with 9 CPUs, 18 GB RAM and user 1 with $\overrightarrow{d_1} = \langle 1/4, 1 \rangle$ and user 2 with $\overrightarrow{d_2} = \langle 1, 1/3 \rangle$. Our BUD algorithm generates $b = \frac{2}{3}$ and the same allocation as DRF.

Table 1 summarizes the properties of different algorithms. ES is simply equal share. EQ tries to equalizes the progresses of all users subject to resource constraints. ES is the baseline of equal sharing. In DRF$^+$, users choose their favorite resource from the interchangeable pool and report their demand on that resource. If the final allocation is worse than equal share, they quit and get the $1/n$ share of the system. FDRF forces users to submit same demand in every interchangeable pool regardless of their efficiency matrices. Given the impossibility results, we choose to sacrifice the strategyproofness in BUD. In practice, we developed tools to measure users' demand and efficiency matrix on the fly, instead of relying on users to report them.

Properties	EQ	ES	DRF$^+$	FDRF	BUD
Sharing Incentive	✗	✓	✗	✓	✓
Envy freeness	✗	✓	✓	✓	✓
Pareto efficiency	✓	✗	✗	✗	✓
Strategyproofness	✗	✓	✗	✓	✗

Table 1: Properties of different algorithms.

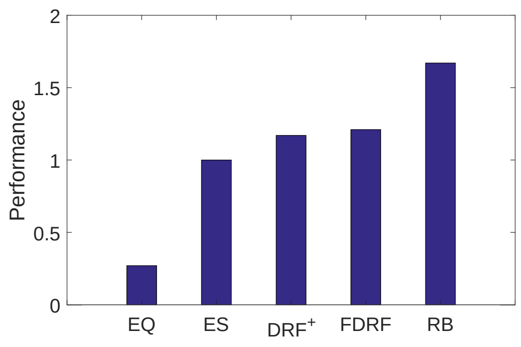

Figure 2: Performance evaluation.

Online allocation. Here the challenge is that users' demands may not be known accurately, yet some estimations can be obtained with error. In traditional resource allocation, we can monitor the progresses of all users periodically and allocate resources to the user with the lowest progress in an online manner to "correct" the imbalance due to estimation errors. However, this strategy does not work with interchangeable resources because progress alone does not reflect the fairness of a resource allocation.

Instead, we propose to use a novel envy graph G where each node represents a user. A directed edge from user-i to user-j means user-i envies user-j, i.e., prefers the allocated resources of user-j. Actually, we can define an envy score $L(i)$ for user i to be its out-degree minus its in-degree in the envy graph G. At each step of the allocation, BUD prioritizes the user with the highest envy score $L(i)$ among those with tasks ready to run. Then the resource allocation and G are updated and again the user with the highest $L(i)$ is picked. This process runs repeatedly.

Numerical simulation. We simulate a cluster environment with 384 CPU cores, 12 GPUs, and 1152 GB memory shared by 4 users. Each user has a different efficiency of using GPU compared to CPU, obtained by running real experiments on a hybrid cluster over Chameleon and Amazon AWS. We compare our BUD algorithm with other algorithms from Table 1 using the total number of jobs finished for a given time as the performance metric. Figure 2 shows normalized performance with respect to ES. Budget-based algorithm outperforms the existing best algorithms for about 30%.

4. REFERENCES

[1] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *NSDI*, 2016.

[2] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.