# UDRF: Multi-resource Fairness for Complex Jobs with Placement Constraints

Yad Tahir*, Shusen Yang†, Alexandros Koliousis*, Julie McCann*
*Imperial College London, †University of Liverpool
{yst11, a.koliousis, j.mccann}@imperial.ac.uk, shusen.yang@liverpool.ac.uk

***Abstract*—In this paper, we study the problem of multi-resource fairness in systems with multiple users. Each user requires to run one or more *complex* jobs that consist of multiple interconnected tasks. A job is considered finished when all its corresponding tasks have been executed in the system. Tasks can have different resource requirements. Because of special demands on particular hardware or software, tasks can have *placement constraints* limiting the type of machines they can run on. We develop *User-Dependence Dominant Resource Fairness (UDRF)*, a generalized version of max-min fairness that combines graph theory and the notion of *dominant resource shares* to ensure multi-resource fairness between users with complex jobs. UDRF satisfies several desirable properties including *strategy proofness*, which ensures that users do not benefit from misreporting their true resource demands. We propose an *offline* algorithm that computes optimal UDRF allocation while the scheduling process can be to be decentralize across multiple schedulers. But optimality comes at a cost, especially for systems where schedulers need to make thousands of online scheduling decisions per second. Therefore, we develop a lightweight *online* algorithm that closely approximates UDRF. Large-scale simulations driven by Google cluster-usage traces show that UDRF achieves better resource utilization and throughput compared to the current state-of-the-art in multi-resource fair allocation.**

## I. Introduction

Cloud computing has become increasingly popular as a cost-effective alternative to proprietary high performance computing systems. But clusters in modern cloud computing environments are almost invariably heterogeneous in terms of their hardware components and software configurations [1]. Such diversity in machines naturally occurs as an organization gradually upgrades its software, adds new machines, decommissions old ones, or enhances others with specialized accelerators like GPGPUs.

Machine heterogeneity brings with it increased complexity associated with user *workflows*. A user may want to run *multiple jobs* in the system. For each job, user may need to access different types (*classes*) of machines to compute a job. As extensively studied in literature [2], [3], a *job* may also consist of several *tasks*. Associating tasks with *placement constraints* (e.g. by using the Condor ClassAds mechanism [4]) is very common in modern datacenters. As recently observed, approximately 50% of jobs at Google have constraints regarding the machines on which they can execute [1]. Placement constraints are not only specific to Google. Such constraints are also supported in modern cluster management systems, including Hadoop YARN [5].

Heterogeneity does not only appear in machines, but also when users require certain resources within a machine class. For instance, data indexing tasks are usually memory-heavy, demanding a relatively large amount of memory compared to other resource types; whereas video transcoding tasks are typically CPU-intensive. Accounting for diversity across machines, users, and their resource demands presents an increased challenge to schedulers for fair provisioning of system components. Getting this right is fundamental to next generation shared computing environments.

Both machine and resource heterogeneity have an impact on the efficient and fair allocation of resources to user workflows. Current resource allocation schemes based on *max-min fairness* like (Hadoop Fair Scheduler [5], Quincy [6] and Seawall [7]) do not deal well with both heterogeneous machine and resource demands. Recently, *multi-resource fairness* schemes [8], [9], [10] such as Dominant Resource Fairness (DRF) become increasingly popular in computation economics. By considering heterogeneous resource demands and capacities, it is proven that these schemes achieve better resource allocation performance than *single-resource allocation* schemes.

However, these approaches assume that the scheduling process is centralized. Thus, using these approaches directly in systems following the master/slave scheduling model is inefficient as the scheduling process cannot be decentralized between multiple slave schedulers. In addition, these approaches fall short when it comes to support users with multiple jobs with heterogeneous complexity, and focus on task-level allocation based on the assumption that users require resources for simple tasks. Utilizing current multi-resource fairness schemes for users with multiple jobs can lead to a substantial waste of resources and inefficient allocation.

In this paper, we present the study of the fair resource allocation problem in computing systems running *complex jobs* that consist of multiple inter-related tasks, all of which need to finish before a job is considered complete. The contributions of this paper are summarized as follows:

- We propose **U**ser-Dependence **D**ominant **Resource** **F**airness (**UDRF**), a new extension to DRF that exploits graph theory, max-min fairness, and dominant resource sharing to ensure multi-resource fairness between users. The newly proposed scheme shows how DRF fairness should be achieved when a user has multiple complex jobs. Each job consists of one or more dependent tasks with strict placement constraints.
- We develop an *offline* algorithm that uses iterative linear

978-1-4799-5952-5/15/$31.00 ©2015 IEEE

programming to compute the optimal UDRF resource allocation. The scheduling process of the offline algorithm can be decentralized to support systems where schedulers follow the master/slave scheduling model. The *offline* algorithm can be expensive when thousands of scheduling decisions have to be made every second. Therefore we propose a lightweight *online* algorithm that approximates the optimal UDRF allocation.

- We prove that UDRF still inherits the four highly desirable properties of DRF including the *sharing incentive* that ensures users on a shared cluster get at least as many resources as the allocation that splits all resources equally between them; *strategy proofness* that avoids unfaithful resource demand reporting; *pareto-efficiency* that ensures no user can increase their allocation without decreasing others; and finally *envy-freeness* that guarantees no user would prefer another's allocation to its own.
- Our experimental results driven by Google cluster traces [11] show that UDRF significantly outperforms the current state-of-the-art in terms of reducing wasted resources while improving job completion times.

The remainder of the paper is organized as follows: We first describe our motivation and system model. The offline and online algorithms are proposed in Section III. Section IV provides theoretical analysis of the UDRF properties. Simulation results are presented in Section V, related work is discussed in Section VI, and conclusions in Section VII.

## II. Motivation and Models

### A. Motivation

To understand the complexity of scheduling resources for a system with multiple concurrent users, consider the scenario shown in Fig. 1 where the system has four machine classes (hadoop, standard, public-IP and high-memory) with different amount of resources (CPU and RAM). The system has four users with heterogeneous resource demands. Note that $u_1$ has two different jobs (*job1* and *job2*) while the rest of the four users has only one single job. In addition, apart from $u_2$, all the jobs of the users consist of one task. The job of $u_2$ has two tasks (*task1* and *task2*).

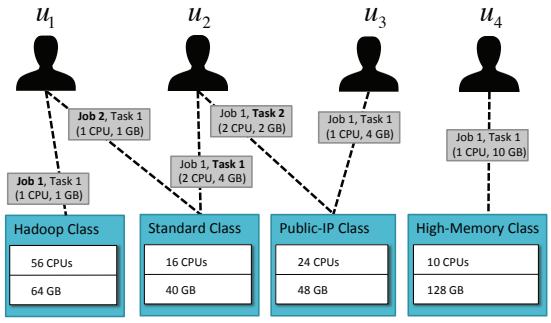



Fig. 1: A system has three machine classes and four users.

A typical approach to apply DRF on the given system is to treat the machine classes as resource types (i.e. the system in this case will have 8 resource types in total). Then, the resource demands of each user is mapped as a *resource demand vector* to all resource types. Fig. 2 shows DRF resource allocation for the system. This allocation is considered to be inefficient as it does not consider the case where the user needs to execute more than one job. Utilizing such abstraction model for resources demands can lead to poor resource allocation. The *dominant resource* of $u_2$ and $u_1$ is the CPU in the standard class and both users get 50% of this resource type. This means that DRF allows $u_1$ to execute 8 jobs of *job2*[1] in the standard class and 8 jobs of *job1* in the hadoop class. This also implies that *job1* consumes only around %12 of the resources available in the hadoop class. The reset of resources are wasted and not utilized by any other user. More efficient resource allocation for $u_1$ would be to increase the resource allocation of *job1* in such away that the dominant resource of *job1* in the hadoop is 50% and the dominant resource of *job2* is also 50%. This will allow $u_1$ to execute 28 jobs of *job1* and 8 jobs of *job2*. $u_1$ benefits from this increase as *job1* and *job2* are completely independent jobs (i.e. there are not two tasks of one job). Furthermore, this increase does not make $u_2$ unfair, as the dominant share for both $u_1$ and $u_2$ is still 50%.

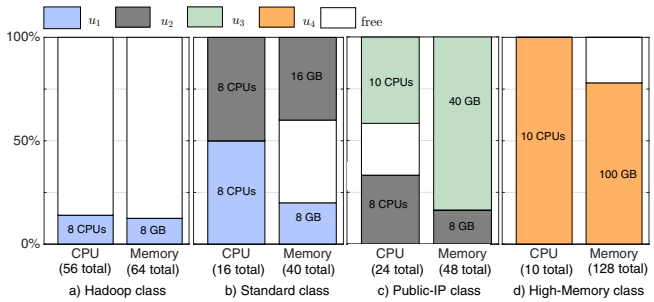



Fig. 2: DRF resource allocation on the system in Fig. 1.

Another important question is that *if the scheduler follows the master/slave model, how can the scheduling process be decentralized?* Herein, scheduling decisions are not made by a *monolithic* scheduler. Instead the system has a *master* resource manager that coordinates a set of *slave* schedulers. It is unclear how users and machine classes should be distributed among the slaves. If it is randomly assigned, tasks may be treated in isolation leading to inefficient resource allocation. For example in Fig. 1, if the resource allocation allows $u_2$ to execute more tasks of *task2* than *task1*, $u_2$ does not benefit from this increase as it does not increase the number of jobs $u_2$ can execute. To allow $u_2$ to run more jobs, the allocation must increase the number of tasks for both *task1* and *task2*. In addition, distributing users arbitrarily may raise fairness issues. Users with common interests on a specific machine class may be handled by multiple schedulers. This may result in users in one slave scheduler getting more shared resources (i.e. become fairer) compared with users handled by other slave schedulers.

### B. System Model

*1) System Capacity:* We consider a networked system consisting of a set of machine classes (MCs) $\mathcal{M}$ and a set of users $\mathcal{U}$. Each machine class provides $K$ types of resources such as CPU, memory, and bandwidth. For $m \in \mathcal{M}$, let

[1]The number of jobs is computed from dividing the amount of dominant resource allocated over the dominant resource demand. For $u_1$, it is $\frac{8}{1} = 8$.

a $K$-dimensional vector $\mathbf{c}_m = (c_{m,1}, ..., c_{m,K})$ to represent its resource capacities, where each entry $c_{m,k}$ $1 \leq k \leq K$ represents the resource capacity of $k$th resource of $m$. The resource capacity of the system $\mathbf{C}$ is represented as a $|\mathcal{M}| \times K$-dimensional matrix.

In the rest of this paper, we will use a tuple $(m, k),\ m \in \mathcal{M}, 1 \leq k \leq K$ to refer to a specific resource.

*2) User Jobs and Resource Demand Graph:* Let $\mathbf{J}^u$ represents the set of jobs that user $u \in \mathcal{U}$ wants to execute in the system. Each job $j \in \mathbf{J}^u$ requires a set of resources from each machine class. Let $\mathbf{R}^j$ be a $|\mathcal{M}| \times K$-dimensional matrix where each entry $\mathbf{R}^j_{m,k}$ is the amount of resource that job $j$ requires from $(m, k)$, which can be either zero or positive. Let $\mathbf{R}^u$ be the *gross resource requirements* of user $u \in \mathcal{U}$ such that

$$\mathbf{R}^u_{m,k} = \sum_{j \in \mathbf{J}^u} \mathbf{R}^j_{m,k} \tag{1}$$

The *strictly positive gross demand* of $u$ is defined as

$$\mathcal{P}_u := \{(m, k) : \mathbf{R}^u_{m,k} > 0, m \in \mathcal{M}, 1 \leq k \leq K\} \tag{2}$$

We defined $\mathcal{U}_{m,k}$ as a set of users that has *strictly positive gross demand* for resources $(m, k)$

$$\mathcal{U}_{m,k} := \{u : \mathbf{R}^u_{m,k} > 0, u \in \mathcal{U}\} \tag{3}$$

### C. Topological User Dependency

To model the dependency of all users in $\mathcal{U}$ in terms of their resource demands, we define the following graph model.

**Definition 1.** [User Dependency Graph]. Let an undirected graph $G(\mathcal{U}, \mathcal{L})$ denotes the dependency of all users, where

$$\mathcal{L} := \{(u, v) : u, v \in \mathcal{U},\ \mathcal{P}_u \cap \mathcal{P}_v \neq \emptyset\}$$

is the set of the edges of the graph. As observed, each edge $(u, v) \in \mathcal{L}$ indicates that two users $u$ and $v$ share one or more resources provided by the system.

**Definition 2.** [Resource-Dependent User Cluster (RDUC)]. For a given user dependence graph $G(\mathcal{U}, \mathcal{L})$, we define a Resource-Dependent User Cluster (RDUC) $\mathcal{C} \subseteq \mathcal{U}$ as a set of users that are in one connected component[2] of $G(\mathcal{U}, \mathcal{L})$. From this definition, two users $u_1$ and $u_2$ may not directly have common resources in a RDUC $\mathcal{C}$. However the resource demands of $u_1$ would certainly affect all the users $v \in \mathcal{U}_{m,k},\ (m, k) \in \mathcal{P}_{u_1}$. Repeating this process for each $v$ realizes that $u_2$ is indirectly dependent on $u_1$. This also concludes that *all users in any given RDUC are either directly or indirectly affect each other*.

### D. Objective and Desired Properties

The objective of this paper is to develop a job scheduling algorithm to find job assignment vector for each user:

$$\mathbf{x}^u = (x_1, x_2, ..., x_{|\mathbf{J}^u|}) \tag{4}$$

[2]A connected component in graph theory is defined as a subgraph that provides a maximal set of vertices such that paths are found between every two vertices.

where $x_j$ represents the number of times for job type $j \in \mathbf{J}^u$ user $u$ can execute in the system. The resource allocated for $u$ user is denoted as a matrix $\mathbf{A}^u(\mathbf{x}^u)$ where each entry is defined as:

$$\mathbf{A}^u_{m,k} = \sum_{j \in \mathbf{J}^u} x_j \mathbf{R}^j_{m,k} \tag{5}$$

**Definition 3.** [Feasible Job Allocation]. *Job assignment vectors for all the users are feasible if the following condition is satisfied*:

$$\sum_{u \in \mathcal{U}} \mathbf{A}^u(\mathbf{x}^u) \preceq \mathbf{C} \tag{6}$$

where $\preceq$ means entry-wise smaller than or equal to.

The scheduling algorithm is expected to achieve the following desirable properties:

- *Pareto-efficiency.* The number of jobs allocated for any user $u \in \mathcal{U}$, $x_u$ cannot be increased without decreasing job allocation of other users $x_v,\ v \in \mathcal{U},\ v \neq u$.
- *Envy-freeness.* No user would want to change their allocation with the allocation of another user:
$$x^u(\mathbf{A}^u) \geq x^u(\mathbf{A}^v),\ \forall u, v \in \mathcal{U}, v \neq u \tag{7}$$
where $x^u(\mathbf{A}^u)$ represents the total number of assigned jobs that uses the dominant demand resource. User $u$ obtains $x^u(\mathbf{A}^u)$ once $u$ is a saturated user.
- *Sharing Incentive.* No user is better off if all resources are equally partitioned among them:
$$x^u(\mathbf{A}^u) \geq x^u(\mathbf{C}/|\mathcal{U}|),\ \forall u \in \mathcal{U} \tag{8}$$
- *Strategy-proofness.* No user can increase their job allocation by lying about their demands, i.e.
$$x^u(\widehat{\mathbf{R}^u}) \leq x^u(\mathbf{R}^u),\ \forall u \in \mathcal{U},\ \forall\ \widehat{\mathbf{R}^u} \neq \mathbf{R}^u \tag{9}$$
where $x^u(\widehat{\mathbf{R}^u})$ represents the number of jobs allocated to user $u$ according to their claimed demand $\widehat{\mathbf{R}^u}$.

## III. User-Dependence Dominant Resource Fairness

Let $d^j_{m,k} = \frac{\mathbf{R}^j_{m,k}}{c_{m,k}}$ be the *job demand share* for job $j$ from resource type $k$ in machine class $m$.

In a RDUC $\mathcal{C} \in \mathbb{C}$, the *dominant user demand share* over all demanded resources $\mathcal{P}(\mathcal{C}) = \bigcup_{u \in \mathcal{C}} \mathcal{P}_u$ is therefore defined as:

$$d^{\max}_u = \max_{(m,k) \in \mathcal{P}(\mathcal{C})} \sum_{j \in \mathbf{J}^u} d^j_{m,k} \tag{10}$$

Given that, the job demand share can be *normalized* with respect to user dominant demand share as follow:

$$\hat{d}^j_{m,k} = \frac{d^j_{m,k}}{d^{\max}_u}, j \in \mathbf{J}^u \tag{11}$$

We define *RDUC-wise dominant share* for a user in a RDUC $\mathcal{C} \in \mathbb{C}$ as:

$$s^{\max}_u = d^{\max}_u \max_{(m,k) \in \mathcal{P}(\mathcal{C})} \sum_{j \in \mathbf{J}^u} x_j \hat{d}^j_{m,k} \tag{12}$$

### A. Offline Algorithm

The main objective of the offline algorithm is to achieve max-min fairness for RDUC-wise dominant share for each RDUC. This is computed while the scheduling process can be decentralized over multiple slave schedulers. The pseudo code of our offline algorithm is summarized in Fig. 3.

**Variables**:
$t$: current round of the algorithm.
$\mathcal{J}_{us}(t)$: the set of all unsaturated jobs at round $t$.
$\mathcal{J}_{s}(t)$: the set of newly saturated jobs at round $t$.
$\mathcal{U}_{us}$: set of unsaturated users.
$\mathcal{U}_{s}$: set of saturated users.

**Input**:
$\mathbf{R}^j_{m,k}$ resource demands for $\forall j \in \mathbf{J}^u, u \in \mathcal{U}$

**Output**:
$\mathbf{X} := \{x_j : \forall j \in \mathbf{J}^u, u \in \mathcal{U}\}$

**Functions**:
***findRDUCs***$(\mathcal{U}, \mathbf{C})$: returns a set of RDUCs for the given users.

**Master - Main Algorithm**:
01: $\mathbb{C} \leftarrow$ ***findRDUCs***$(\mathcal{U}, \mathbf{C})$; // update RDUCs
02: **for all** $\mathcal{C} \in \mathbb{C}$ **do**
03: $\mathbf{R}(\mathcal{C}) \leftarrow \{\mathbf{R}^j_{m,k} : u \in \mathbb{C},\ j \in \mathbf{J}^u,\ (m,k) \in \mathcal{P}(\mathcal{C})\}$; //construct all the demanded resources for $\mathcal{C}$
04: $[\ \mathbf{X}\ ] \leftarrow$ **scheduleJobs**$(\mathcal{C}, \mathbf{R}(\mathcal{C}), \mathbf{C}, \mathbf{X})$;
05: **end for**

**Slave - scheduleJobs**$(\mathcal{C}, \mathbf{R}(\mathcal{C}), \mathbf{C}, \mathbf{X})$:
06: $\mathcal{U}_{us} \leftarrow \{u : u \in \mathcal{C}\}, \mathbf{C}(t) \leftarrow \mathbf{C}$;
07: **while** $\mathcal{U}_{us} \neq \emptyset$ **do**
08: $\mathcal{J}_{us}(t) \leftarrow \{j : j \in \mathbf{J}^u, u \in \mathcal{U}_{us}\}$; // update unsaturated jobs
09: **compute** $\hat{d}^j_{m,k}, \forall j \in \mathcal{J}_{us}(t)$; //based on Eq. 11
10: $b \leftarrow 1/\max\limits_{(m,k)\in\mathcal{P}(\mathcal{C})} \sum\limits_{j\in\mathcal{J}_{us}(t)} \hat{d}^j_{m,k}$; //RDUC-dominant share maximization
11: **while** $\mathcal{J}_{us}(t) \neq \emptyset$ **do**
12: **for all** $u$ has a job $j \in \mathcal{J}_{us}(t)$
13: $\mathcal{S}(t) \leftarrow \{(m,k) \in \mathcal{P}(\mathcal{C}), c_{m,k}(t) \neq 0\}$;
14: $r_u \leftarrow \max\limits_{(m,k)\in\mathcal{S}(t)} \sum\limits_{j\in\mathbf{J}^u} \hat{d}^j_{m,k}$;
15: $\hat{x}_j \leftarrow \frac{b}{r_u d^{\max}_u}, \forall j \in \{v : v \in \mathbf{J}^u, v \in \mathcal{J}_{us}(t)\}$; // allocate jobs
16: $\hat{\mathbf{A}^u} \leftarrow \mathbf{A}^u$, **compute new** $\mathbf{A}^u(\hat{x}_j)$; //based on Eq. 5
17: **end for**
18: $\mathbf{C}(t) \leftarrow \mathbf{C} - \sum\limits_{j\in\mathbf{J}^u} \hat{x}_j \mathbf{R}^j_{m,k}, \forall u \in \mathcal{U}_{us}$; //update capacity
19: $\mathcal{J}_s(t) \leftarrow \{j : \exists (m,k) \in \mathbf{R}^j > 0, c_{m,k}(t) = 0 \vee \hat{\mathbf{A}^u} = \mathbf{A}^u\}$;
20: $\mathcal{J}_{us}(t+1) \leftarrow \mathcal{J}_{us}(t+1) - \mathcal{J}_s(t)$; //new unsaturated jobs
21: $t \leftarrow t+1$;
22: **end while**
23: $x_j \leftarrow \hat{x}_j, \forall j \in \mathbf{J}^u, u \in \mathcal{U}_{us}$;
24: $\mathbf{C} \leftarrow \mathbf{C}(t)$;
25: $\mathcal{U}_s \leftarrow \{u : \exists (m,k) \in \mathcal{P}_u, c(m,k) = 0\}$;
26: $\mathcal{U}_{us} \leftarrow \mathcal{U}_{us} - \mathcal{U}_s$;
27: **end while**
28: **return X**;

Fig. 3: The UDRF offline algorithm.

In the beginning of the algorithm, the master scheduler first computes $\mathbb{C}$ (line 01). Here, the function ***findRDUCs***$(\mathcal{U}, \mathbf{C})$ establishes the user dependence graph $G(\mathcal{U}, \mathcal{L}(t))$ at round $t$, by setting $\mathcal{L}(t) \leftarrow \{(u,v) : u, v \in \mathcal{U}, \mathcal{P}_u \cap \mathcal{P}_v \neq \emptyset\}$ and then it computes $\mathbb{C}$ by detecting all graph components of $G(\mathcal{U}, \mathcal{L}(t))$, based on graph traversal algorithms (e.g. Breadth-First Search) with linear complexity of $O(|\mathcal{U}| + |\mathcal{L}(t)|)$ [12].

After the completion of $\mathbb{C}$, in line 04 the master scheduler allocates each graph component $\mathcal{C} \in \mathbb{C}$ to a slave scheduler. The slave scheduler then independently executes **scheduleJobs**$(\mathcal{C}, \mathbf{R}(\mathcal{C}), \mathbf{C}, \mathbf{X})$ and returns the number of job allocated for each user. It is worth noting that in this approach slave schedulers do not need to communicate with each other during the scheduling process. If the system does not adhere to the master/slave paradigm (i.e. the system has one scheduler), then the master scheduler computes **scheduleJobs** by itself.

From line 06-28, the offline algorithm maximizes the minimum level of RDUC-wise dominant share for each user in the RDUC. In line 08-10, the offline algorithm equalizes the RDUC-wise dominant share that each user get from the RDUC. This is done through multiple iterations. In each iteration, the algorithm first adds the jobs of the unsaturated users to the unsaturated job set (line 08). Then the algorithm finds the job assignment vector for each unsaturated user through one or more iterations (line 11-22). In each iteration, the algorithm increases job allocation for each unsaturated job (line 14-15) while the constraint defined in Eq. 12 is strongly maintained. Line 14 finds a modifier that equalizes the dominant share of all jobs belonging a user to $d^{\max}_u$. After resources are allocated (line 16) to all current unsaturated jobs $\mathcal{J}_{us}(t)$, the algorithm updates the resource capacity (line 18) and removes the jobs that has been saturated (line 19-20). We define a saturated job as a job that either has a positive demand for resource type but the resource is saturated, or the algorithm cannot increase its allocation anymore because of constraint 12. In line 25-26, we remove the users which has at least a job depending on one or more saturated resource from the unsaturated user set. This process is repeated until all users in the RDUC are saturated.

### B. Numerical example

Consider the four-user example in Section II-A. In the beginning, the ***findRDUCs*** function returns two RDUCs ($\mathcal{C}_1$ and $\mathcal{C}_2$). $\mathcal{C}_1$ contains three users $u_1$, $u_2$, and $u_3$ and $\mathcal{C}_2$ has only one user $u_4$. At the first round, the offline algorithm allocates resources for users in $\mathcal{C}_1$. As result, it is going to add the jobs of $u_1$, $u_2$, and $u_3$ to the unsaturated job set $\mathcal{J}_{us}$. The offline algorithm equalizes the dominant shares of $u_1$, $u_2$, and $u_3$ as they belong to the same RDUC ($\mathcal{C}_1$). The resource shares for $u_1$ and $u_2$ in the standard class are $\langle 1/2,\ 2/5\rangle$ and $\langle 1/2,\ 1/5\rangle$ respectively. In the hadoop class $u_1$ gets $\langle 1/7, 1/8\rangle$. For the resources in the public-IP class, $u_2$ gets $\langle 1/3,\ 1/6\rangle$ and $u_3$ receives $\langle 1/4,\ 1/2\rangle$. This round saturates all the jobs except to *job1* for $u_1$ and *job1* to $u_3$. Currently, $d^{\max}$ for $u_1$, $u_2$, and $u_3$ in this allocation are equal to $1/2$. In the next iteration, the algorithm allocates more resources for unsaturated jobs.

The new resource allocation for $u_1$ in the hadoop class is $\langle 1/2, 7/16\rangle$. Similarly, in the public-ip class $u_3$ allocation is $\langle 5/12, 5/6\rangle$. In the next round, the algorithm will process the users in $\mathcal{C}_2$. Since $u_4$ is the only active user in $\mathcal{C}_2$, UDRF allows the job of $u_4$ to have the resource shares $\langle 1, 25/32\rangle$. Fig. 4 shows the final outputs of the offline algorithm.

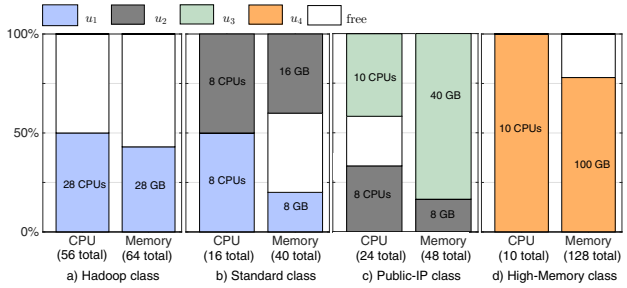



Fig. 4: UDRF allocation for the scenario shown in Fig. 1.

### C. Online Algorithm

In cloud computing, resource demands for machine classes can be highly dynamic. New users join the cluster, other users depart from it. Users may change their resource requirements. Datacenter schedulers may require making thousands of online decisions in every second as new jobs continuously arrive or when resources become available. The offline algorithm is expensive to compute. Therefore, the online algorithm seeks an approximation of UDRF that can be efficiently implemented in todays online schedulers. Fig. 5 outlines how schedulers based on UDRF can allocate resources to new incoming jobs without modifying the allocation of existing ones.

1: **if** a new user comes or an existing user leaves the system **then**
2: update the user dependence graph and corresponding RDUCs;
3: allocate each RDUC to a slave;
4: **end if**
5: **If** a resource in a RDUC $\mathcal{C}$ frees up **then**
6: the corresponding scheduler for RDUC allocates a job for user in $\mathcal{C}$ with lowest RDUC-wise dominant share without breaking the feasibility of the current allocation;
7: **end if**

Fig. 5: The UDRF online algorithm.

## IV. Fairness Properties Achieved by UDRF

This section demonstrates how UDRF achieves the fairness properties of Pareto-efficiency, Strategy-proofness, Envy-freeness, and Sharing Incentivisation.

**Lemma 1.** Each user $u \in \mathcal{U}$ in UDRF allocation has at least one saturated resource at a MC

$$\exists (m,k) \in \mathcal{P}_u, \ \textbf{\textit{s.t.}} \sum_{v \in \mathcal{U}_{m,k}} \mathbf{A}^v_{m,k} = c_{m,k} \tag{13}$$

**Proof.** We proof this lemma by contradiction. Suppose there exists a user $u$ such that all resources are unsaturated under the UDRF allocation,

$$\forall (m,k) \in \mathcal{P}_u, \ \textbf{\textit{s.t.}} \sum_{v \in \mathcal{U}_m} \mathbf{A}^v_{m,k} < c_{m,k} \tag{14}$$

This means that $u$ is not a saturated user, which implies that the offline algorithm does not terminate and UDRF allocation has not yet been computed. This contradicts the fact that $u$ is under the UDRF allocation. □

**Theorem 1.** UDRF achieves Pareto-efficiency.

**Proof.** According to Lemma 1, each user $u$ has a saturated resource $(m,k)$. We have

$$\sum_{u \in \mathcal{U}_{m,k}} \sum_{j \in \mathbf{J}^u} x_j \mathbf{R}^j_{m,k} = c_{m,k} \tag{15}$$

If no other user uses the saturated resource $(m,k)$ (i.e. $\sum_{j \in \mathbf{J}^u} x_j \mathbf{R}^j_{m,k} = \mathbf{A}^u_{m,k} = c_{m,k}$), then $x_u$ cannot be increased, since $c_{m,k}/\mathbf{R}^u_{m,k}$ is fixed; otherwise, the increase of $x_u$ must result in the decrease of some other users $v \in \mathcal{U}_{m,k}$ that also use this resource, according to (15). □

**Theorem 2.** UDRF allocation is envy-free.

**Proof.** Consider two users $u$ and $v$. Let $t_u$ and $t_v$ be the last rounds in which $u$ and $v$ were allocated resources. We have two cases:

- If $t_v > t_u$, then $u$ became saturated earlier than $v$. This implies that $\mathcal{P}_u \not\subseteq \mathcal{P}_v$ and $x^u(\mathbf{A}^v) = 0$.
- If $t_v \leq t_u$, then it is obvious that $s^{\max}_u \geq s^{\max}_v$. Therefore $x^u(\mathbf{A}^u) = s^{\max}_u / d^{\max}_u \geq s^{\max}_v / \max d^{\max}_u = x^u(\mathbf{A}^v)$.

Both cases demonstrate $x^u(\mathbf{A}^u) \geq x^u(\mathbf{A}^v)$ , i.e. $u$ does not envy $v$. □

**Theorem 3.** UDRF allocation achieves sharing incentive.

**Proof.** Let $t$ be the last round in which a user $u$ were allocated resources (i.e. $u$ is a newly saturated user at round $t$). Let $\mathcal{C}$ be the RDUC to which $u$ belongs. Its dominant share

$$s^{\max}_u(t) \quad \geq \quad 1/|\mathcal{C}| \geq 1/|\mathcal{U}|$$

Let $(m,k)$ be the dominant resource of $u$ (i.e. $\sum_{j=1}^{|\mathbf{J}^u|} d^j_{m,k}(t) = d^{\max}_u(t)$), we have :

$$\begin{aligned} x^u(\mathbf{A}^u) &= \frac{1}{\mathbf{R}^u_{m,k}} \mathbf{A}^u_{m,k}(t) = \frac{1}{\mathbf{R}^u_{m,k}} \frac{s^{\max}_u(t)}{\sum_{j=1}^{|\mathbf{J}^u|} d^j_{m,k}(t)} \mathbf{R}^u_{m,k} \\ &\geq \frac{1}{\mathbf{R}^u_{m,k}} \frac{1}{|\mathcal{U}|} \frac{c_{m,k}}{\mathbf{R}_{m,k}} \mathbf{R}^u_{m,k} \geq \frac{1}{\mathbf{R}^u_{m,k}} \frac{c_{m,k}}{|\mathcal{U}|} \\ &= x^u(\mathbf{C}/|\mathcal{U}|) \end{aligned}$$

This completes the proof of Theorem 3. □

**Theorem 4.** UDRF allocation is strategy proof.

**Proof.** Consider a user $u$ reports an untruthful resource demand matrix $\widehat{\mathbf{R}}^u \neq \mathbf{R}^u$. Let $t$ and $\widehat{t}$ be the rounds in which $u$ was saturated for the faithful and unfaithful reports respectively. Suppose $x^u(\widehat{\mathbf{R}}^u) > x^u(\mathbf{R}^u)$, which means $\forall (m,k) \in \mathcal{P}_u$, $\mathbf{A}^u_{m,k} < \widehat{\mathbf{A}}^u_{m,k}$ and $t < \widehat{t}$, where $\widehat{\mathbf{A}}^u_{m,k}$ represents the allocation of a given resource $(m,k)$ when $u$ claims $\widehat{\mathbf{R}}^u$. Let $(m^*,k^*)$ be the saturated resource (at round $t$) when $u$ claims $\mathbf{R}^u$). We have two cases

1) If $\{u\} = \mathcal{U}_{m^*,k^*}$, $(m^*,k^*)$ has only one user $u$. Therefore, we have

$$\mathbf{A}^u_{m^*,k^*} = c_{m^*,k^*} \geq \widehat{\mathbf{A}}^u_{m^*,k^*} \tag{16}$$

2) If $\{u\} \subset \mathcal{U}_{m^*,k^*}$. Since $t < \widehat{t}$, $(m^*, k^*)$ should not be saturated at $t$ when $u$ claims $\widehat{\mathbf{R}}^u$, which implies that

$$\widehat{\mathbf{A}}^u_{m^*,k^*} \leq c_{m^*,k^*} - \sum_{v \in \mathcal{U}_{m^*,k^*} - \{u\}} \mathbf{A}^v_{m^*,k^*} = \mathbf{A}^u_{m^*,k^*}$$

Both cases contradict the supposition that $\mathbf{A}^u_{m,k} < \widehat{\mathbf{A}}^u_{m,k},\ \forall (m,k) \in \mathcal{P}_u$. □

## V. Evaluation

We evaluated UDRF against the current state-of-the art in multi-resource fairness, DRF. We use Google workload traces published in [11]. The traces contain task scheduling information for a 12K-machine cluster. For confidentiality, the resource capacities of machines in the traces are normalized so that the value of the most powerful machine is 1. Each machine has one or more attributes, which are key-value pairs representing machine properties (kernel version, clock speed, external IP, etc). Users represent Google engineers and services who submit jobs to the cluster. A job is comprised of one or multiple tasks, each of which is accompanied by a set of resource requirements and constraints on machine attributes. From the Google traces we extract the computing demands, placement constraints, and job arrival times for jobs and its corresponding tasks. Based on resource capacities and machine attributes, we assign each machine into its corresponding machine class. In total, the cluster has 700 distinct classes. We use the placement constraints of the tasks to map the dependencies between the users and the machine classes. To simulate that a user may want execute a job multiple times, we randomly assign each job with a variable which indicates the total number of times the job has to run in the system.

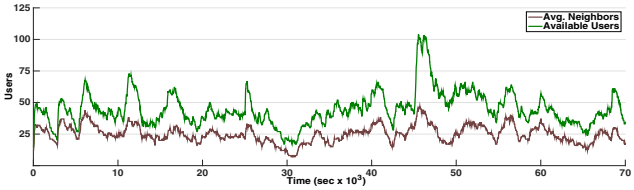



Fig. 6: Number of users with average user degree in RDUCs.

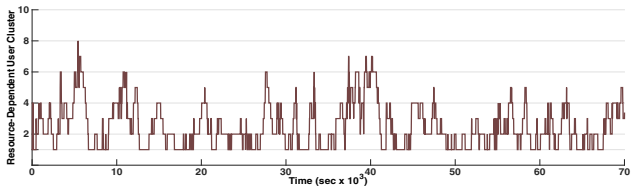



Fig. 7: Number of RDUCs existing in the system.

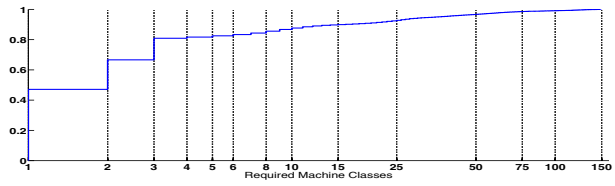



Fig. 8: CDF of required machines classes for the jobs.

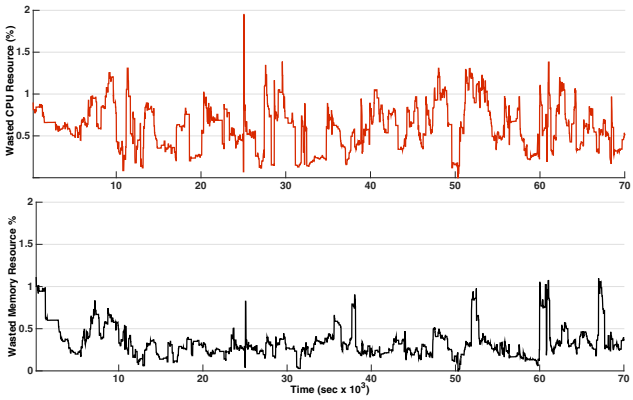
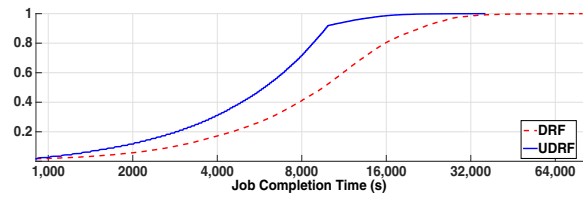


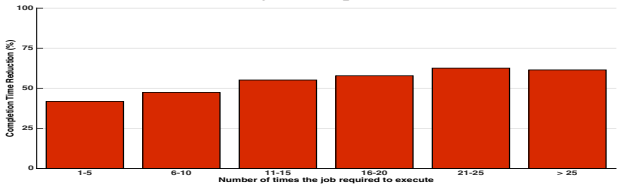

Fig. 9: Wasted CPU and Memory from DRF in compare to UDRF.

Overall, the 24-hour data contains information about 580 users and 19K jobs. Fig. 6 depicts that the number of users using the system is time varying. We observe that the more users are available in the system, the higher the node degree in RDUCs. This also has an effect on the number of RDUCs existing in the system, as depicted in Fig. 7. Throughout the experiment, the system can have from one to eight RDUCs. We notice that around 65% of jobs required between one to two machine classes as shown in Fig. 8 where the CDF of the required machine classes of the jobs is displayed. Users requesting to execute a large number of jobs often have demands on a wide range of machine classes.

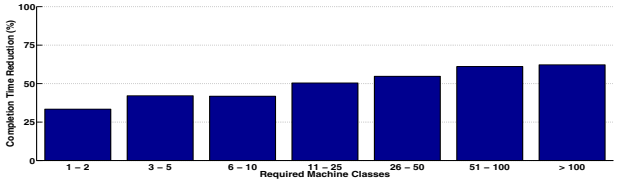
We compare the performance of UDRF with respect to scheduling with DRF. We have implemented two schedulers. The first scheduler uses the online algorithm of UDRF, and the second scheduler runs the *progressive filling* of DRF [8]. In the DRF scheduler, the machine classes are mapped as resources as described in Section II-A. Resource scheduling decisions




(a) CDF of job completion times.




(b) Completion time reduction for jobs with different execution times requirements




(c) Completion time reduction categorized by machine classes.

Fig. 10: UDRF improvements on job completion times over DRF.

are made on a per-second basis. Both schedulers have fixed-size waiting queues with a capacity of 100 jobs. Whenever the waiting queue is not full, we add jobs to the queue until it becomes saturated. To reduce the simulation time, job running times are randomly assigned between 10 and 100 seconds. When a running job is executed in the system for the requested number of times, it will be removed from the waiting queue.

Fig. 9 depicts the time series of wasted CPU and memory resources in the cluster running DRF. The inefficiency in DRF resource allocation translates into longer job completion times for the jobs. Fig. 10a shows the CDFs of job completion times of both UDRF and DRF. From Fig. 10c, it can be seen that the more machine classes the job needs, the more time reduction is expected. Similar pattern is also observed in job with different execution requested times. As shown in Fig. 10b, the more times the job is required to execute, the higher reduction in completion time is expected from UDRF in compare to UDF.

The primary reason causing wasted resources is that DRF treats a user with multiple jobs as task-level resource allocation. In this case, the number of job instance of each job are equal and fixed. This causes DRF to do not utilizes the all resource available and not used by other users. This also means that jobs with high requirements on the number of times to execute will take longer time to finish and leave the waiting queue. The more jobs stay longer in the waiting queue, the higher delay is expected for the newly arrived jobs.

## VI. Related Work

Max-min fairness is one of the most well-known fairness schemes used in datacenters and cloud. Over the years, many extensions of max-min fairness have been proposed ranging from priority, proportional sharing, and strict placement constraints [13]. One approach to apply max-min in multi-resource systems is to employ *single-resource abstractions* where system resources are spitted into fixed partitions, commonly known as *slots*. Then allocations are performed at the granularity of these slots. Using such simple abstractions fall short when resource capacities and user resource demands are heterogeneous [8]. Ghodsi *et al.* [8] propose Dominant Resource Fairness (DRF) as an alternative approach to achieve resource fairness in multi-resource systems. DRF satisfies several highly desirable fairness properties, and it quickly received significant attentions from from academia and industry. Parkes *et al.* [9] extend the DRF scheme to provide a attractive solution for weighted users and users with zero resource demands. Moreover, Wang *et al.* [10] propose the notion of *global dominant share* to address the heterogeneity of server capabilities in cloud computing.

All the approaches above focus on systems running simple, independent tasks, while we focus on allocating resources for clusters for users running complex jobs that consist of several tasks. Herein, users can have multiple jobs and tasks are dependent and precedence relationships exist between them. Also, none of the existing work address how DRF fairness scheme can be applied for systems running the master/slave scheduling model. The work in [2], [3] are considered to be closest to our work as they propose scheduling for complex workflows. However, the mentioned solutions are limited to single-resource allocation and none of them focus on the four attractive (yet important) properties that UDRF satisfies.

## VII. Conclusion

This paper addresses multi-resource allocation problem in systems where users want to run complex jobs. A job is composed of one task or more, all of which have to complete before a job is considered as finished. In this paper, we explain why the current state-of-the-art falls short when it comes to achieve multi-resource fairness between users with one or more complex jobs. We propose *User Dependence Dominant Resource Fairness (UDRF)* as a solution to this problem. We analyze UDRF and show that it satisfies several highly desirable fairness properties. We propose two algorithms (*offline* and *online*) to find UDRF allocation. We also illustrate how the UDRF scheduling process can be distributed across multiple schedulers in systems adhering to the master/slave scheduling model. Our simulations driven show that, compared to the current state-of-the-art in multi-resource fairness, UDRF achieves significant improvements in terms of reducing wasted resources while improving job completion times.

## VIII. Acknowledgements

This research was supported by the Ministry of Higher Education, Kurdistan, Iraq and the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318521.

## References

[1] B. Sharma, V. Chudnovsky, J. Hellerstein, R. Rifaat, and C. Das, "Modeling and synthesizing task placement constraints in Google compute clusters," in *Proc. ACM SoCC*, 2011.

[2] F. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. ACM SIGCOMM*, 2014.

[3] M. Chowdhury, M. Zaharia, J. Ma, M. Jordan, and I. Stoica, "Managing data transfers in computer clusters with Orchestra," in *Proc. ACM SIGCOMM*, 2011.

[4] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proc. ACM HPDC*, 1998.

[5] "Apache Hadoop NextGen MapReduce." [Online]. Available: http://hadoop.apache.org

[6] M. Isard, V. Prabhakaran, and J. Currey, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. ACM SOSP*, 2009.

[7] A. Shieh, S. Kandula, V. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. USENIX NSDI*, 2011.

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

[9] D. Parkes, A. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," in *Proc. ACM EC*, 2012.

[10] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proc. IEEE INFOCOM*, 2014.

[11] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces." [Online]. Available: http://code.google.com/p/googleclusterdata/

[12] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Communications of the ACM*, vol. 16, no. 6, pp. 372–378, 1973.

[13] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Proc. ACM EuroSys*, 2013.