Multi-resource fair sharing for multiclass workflows

Jian Tan, Li Zhang, Min Li, Yandong Wang
IBM T. J. Watson Research Center, NY 10598, USA
tanji@us.ibm.com, zhangli@us.ibm.com, minli@us.ibm.com, yandong@us.ibm.com

ABSTRACT

Multi-resource sharing for concurrent workflows necessitates a fairness criteria to allocate multiple resources to workflows with heterogeneous demands. Recently, this problem has attracted increasing attention and has been investigated by assuming that each workflow has a single class of jobs and that each class contains jobs of the same demand profile. The demand profile of a class represents the required multi-resources of a job. However, for typical applications in cloud computing and distributed data processing systems, a workflow usually needs to process multiple classes of jobs. Relying on the concept of slowdown, we characterize fairness for multi-resource sharing and address scheduling for multiclass workflows. We optimize the mixture of different classes of jobs for a workflow as optimal operation points to achieve the least slowdown, and discuss desirable properties for these operation points. These studies assume that the jobs are infinitely divisible.

When jobs are non-preemptive and indivisible, any fairness criteria that only relies on the instantaneous resource allocation cannot be strictly maintained at every time point. To this end, we relax the instantaneous fairness to an average metric within a time interval. This relaxation introduces a time average to fairness and allows occasional, but not too often, violations of instantaneous fairness. In addition, it brings flexibility and opportunities for further optimization on resource utilization, e.g., using bin-packing, within the constraint on fairness.

1. INTRODUCTION

Multi-resource fair allocation is a fundamental problem in designing computing systems shared by multiple workflows. A workflow consists of multiple classes of jobs, and each class contains multiple jobs. This general definition describes many applications. For example, a MapReduce [1] job can be viewed as a workflow. It contains two classes of jobs: map tasks and reduce tasks. Dryad [12] and Spark [2] can support more than two classes of jobs. These applications in cloud computing and distributed data processing systems rely on efficient allocation of multiple resources, e.g., CPU time, memory space, I/O bandwidth, network transmission, software licenses, to name just a few. This practical requirements foster the research on fair sharing of multiple resource types [8, 13, 17, 14, 5, 4].

In a degenerate case when all workflows consume a single type of resource, fairness has been well studied [21, 11], primarily due to the complete preference order induced by the fraction of the resource used for each running workflow. Recently, the problem on fair sharing of multiple resources among multiple workflows has attracted much attention [8, 13, 17, 14, 4]. These studies assume that a workflow contains multiple identical jobs and that each job demands a certain amount of resource for each of the multiple types of resources. Therefore, a job class can be characterized by

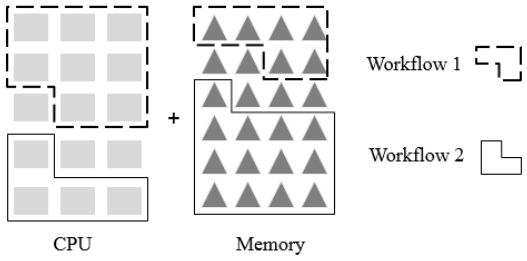

Figure 1: Multi-resource requirement of 2 workflows

a demand profile, as a vector described by the required units of each type of resource for a unit of job. As illustrated in Fig. 1, a job of workflow 1 uses 8 CPUs and 6 memories, represented by a profile $(8, 6)$. Similarly, the job profile of workflow 2 is $(4, 13)$.

Because of heterogeneous demands, it is quite common that some jobs require more CPU than memory and others require more I/O bandwidth than CPU. If each job class only consumes a single type of resource, then the workflow's resource usage can be represented by a scalar, which implies a complete preference order. However, with multiple resources, the resource requirements are vectors according to the demand profiles, allowing only a partial order. Therefore, a fairness criteria is needed to schedule multiple workflows on a shared computing cluster. Specifically, in a single resource setting, most scheduling policies satisfy the work-conserving property, in the sense that all resources are fully utilized when there are unfinished workloads. For the multi-resource setting, due to job profiles that have proportional

resource requirements for different types, it is possible that some resources have leftover that cannot be utilized at all. For example, only two jobs of profile $(8, 6)$ can be served on a cluster with capacity $(16, 30)$. It will have $30 - 2 \times 6 = 18$ units of leftover for the second type of resource. Therefore, fairness can impact system efficiency. To this purpose, a fairness criteria called dominant resource fairness [8] has been proposed, which is to allocate resources on the dominant shares according to max-min fairness. It has been shown [8, 17, 13, 14] that this fairness has a number of properties, such as sharing incentive, envy free, Pareto optimal and strategy proof.

However, for many real applications, e.g., MapReduce [1], Dryad [12], Spark [2], a workflow usually contains multiple classes of jobs, where each class has a different demand profile for resources. For example, a MapReduce job consists of both map tasks and reduce tasks. It is possible that a map task is CPU intensive and a reduce task is I/O intensive. In addition, in some applications different classes of jobs of the same workflow can be executed in parallel with arbitrary combinations, and in other cases there are restrictions that are caused by job dependency constraints so that only specific combinations of different job classes are possible. For all of these applications, it is important to study the fairness and the optimal mixing of different classes of jobs since a workflow cannot be simply characterized by a single demand profile as assumed in [8, 13, 14]. The existing frameworks, e.g., dominant resource fairness [8, 13], and proportional fairness [4], are thus not directly applicable for this setting.

In this study, we distinguish two different cases: 1) jobs are infinitely divisible and, 2) jobs are indivisible and non-preemptive. For the first case, if the allocated resources are not enough for an entire job, a fraction of the job can be launched. This is the assumption made in [8, 13, 14, 4]. For the second case, every job has to be launched in its entirety, and the job keeps running until finish. During the execution of the job, the occupied resources will not be used by any other workflows. For the first case, we consider instantaneous fairness that needs to be satisfied at each time point. This criteria only depends on the resource allocation at that specific time. Thus, we do not need to consider workflow arrivals and departures. However, for the second case, since jobs are non-preemptive and indivisible, instantaneous fairness cannot be guaranteed. We relax the instantaneous fairness to an average measurement on a time interval. This relaxation introduces a time average to fairness and allows occasional, but not too often, violations of instantaneous fairness. In addition, it brings flexibility and opportunities for further optimization on resource utilization, e.g., using bin-packing, within the constraint on fairness.

2. ILLUSTRATIVE EXAMPLE

To illustrate the problem, consider a system of 10 CPUs and 1000GB memory with two workflows A and B. A has two classes of jobs. Each class can have multiple jobs. A job of class A_1 requires 1 CPU and 125GB memory, and a job of class A_2 requires 5 CPUs and 80GB memory. Thus, A has two job profiles $(1, 125)$ and $(5, 80)$. B has three classes of jobs, with B_1 requiring 2 CPU and 50GB memory for each job, B_2 requiring 4 CPUs and 25GB memory for each job, and B_3 requiring 6 CPUs and 80GB memory for each job. It has three job profiles $(2, 50)$, $(4, 25)$ and $(6, 80)$.

We first review dominant resource fairness [8] by assuming that workflows A and B only has a single class, i.e., A_1 and B_1 with job profiles $(1, 125)$ and $(2, 50)$, respectively. In addition, we assume that the jobs are infinitely divisible. After calculations, the dominant resource fairness allocates the limited resources by launching 40/9 jobs for A and 25/9 jobs for B. We use a notion on slowdown of a workflow to evaluate the allocation process from a slightly different perspective. By this notion, we interpret dominant resource fairness in a way that can be generalized to the setting when jobs are non-preemptive and indivisible.

Suppose that the whole cluster is dedicated to serve A. Then $\min\{10/1, 1000/125\} = 8$ jobs of type A_1 can be launched. By the same reason, $\min\{10/2, 1000/50\} = 5$ jobs of type B_1 can be launched for workflow B. Because of sharing, only $N_A \leq 8$ and $N_B \leq 5$ numbers of jobs can be launched for A and B, respectively. The slowdown of workflow A is equal to $N_A/8$ and the slowdown of workflow B is equal to $N_B/5$. A resource allocation is said to be fair for A and B, if their slowdowns are equal, i.e., $\eta = N_A/8 = N_B/5$. The goal is to fully utilize the available resources, thus,

$$\begin{aligned} \max \quad & \eta \\ \text{subject to} \quad & N_A \times 1 + N_B \times 2 \leq 10, \qquad \text{for CPU} \\ & N_A \times 125 + N_B \times 50 \leq 1000, \quad \text{for memory} \\ & \eta = N_A/8 = N_B/5, \quad \text{for fairness} \end{aligned}$$

which yields

$$\begin{aligned} \eta &= \min\{10/\left(8 \times 1 + 5 \times 2\right), 1000/\left(8 \times 125 + 5 \times 50\right)\} \\ &= 5/9. \end{aligned}$$

This result implies that $N_A = 8 \times 5/9 = 40/9$ and $N_B = 5 \times 5/9 = 25/9$. In this case, CPU is fully utilized.

The problem becomes more complicated when A and B have multiple classes of jobs of different profiles. In order to fairly allocate resources to multi-class workflows, we choose to generalize the notion of slowdown for this setting. Then, equalizing the slowdowns of different workflows yields a fair allocation. If we specify the slowdowns of all jobs to be equal, it is possible that there are still leftover resources of some types while other types are saturated. This problem has been discussed in [17]. In this case, if a class does not need these saturated types of resources, we can further increase the number of running jobs of this class, which will eventually make the slowdowns to be unequal. To address this issue, we can either make an assumption that every job class requires all types of resources, or allow multiple rounds of job allocation. In each round, we only consider those types of classes that can be further increased, constrained by fairness and the available leftover resources.

Consider the situation when the whole cluster is dedicated to serve workflow A, with job profiles A_1 and A_2 as described earlier. The number of jobs that can be supported for serving workflow A form a feasible set $S_A = \{(x, y) : x(1, 125) + y(5, 80) \leq (10, 1000), x \geq 0, y \geq 0\}$, where x and y denote the number of running jobs of type A_1 and A_2, respectively. The boundary subset $P_A = \{(x, y) : (x, y) \in S_A$ and there is no $w \in S_A$ with $w > (x, y)\}$ defines a Pareto curve where the capacity constraint is satisfied. The execution speed of the workflow A increases by $x_c/x = y_c/y$ times when the operation point (x, y) proportionally increases to a point $(x_c, y_c) \in P_A$. The slowdown of operation point (x, y) can be defined by x/x_c, which is equal to y/y_c. The same

arguments can be applied for other workflows, e.g., B, which has three types of job profiles, B_1, B_2, B_3.

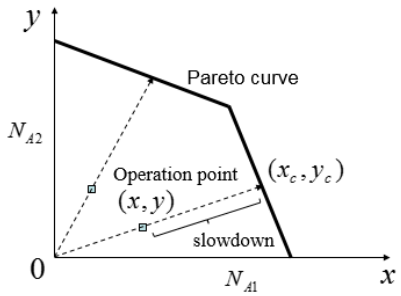

Figure 2: Slowdown with 2 classes of jobs

When jobs are indivisible, the numbers of running jobs of every class have to take integer values. In this case, we can still map an operation point $(x, y), x, y \in N^+$ to a point $(x_c, y_c), x_c, y_c \in N^+$ within the feasible set, such that $x/x_c = y/y_c$ and there exists no other feasible point (x'_c, y'_c) with $x'_c > x_c, y'_c > y_c$. Thus, the point (x_c, y_c) may not lie on the Pareto curve. The slowdown can be defined by the ratio between these two points, similar to the divisible case.

3. MODEL DESCRIPTION

Assume that a job of class $j, 1 \leq j \leq J$ belonging to workflow $i, 1 \leq i \leq I$ requires $R_{ij}(r)$ of resource type r. In the rest of the paper, we use A^T to denote the transpose of a matrix A. Thus, the demand profile of a class j of workflow i can be described by a vector $R_{ij} = (R_{ij}(1), \cdots, R_{ij}(r))^T$. Let N_{ij} denote the number of jobs allocated to class j of workflow i, which is determined by the scheduling algorithm at the run time. We call $\{N_{ij}\}$ the operation point of workflow i. This operation point moves dynamically when the other workflows that share with i changes.

Denote by C_r the capacity of resource r. We have the following constraints, for all r,

$$\sum_i \sum_j R_{ij}(r) N_{ij} \leq C_r.$$

Let $R_i = (R_{i1}, \cdots, R_{iJ})$ be a $I \times J$ matrix, and $N_i = (N_{i1}, \cdots, N_{iJ})^T$, $C = (C_1, \cdots, C_r)^T$ be two vectors. We have

$$\sum_{i=1}^{I} R_i \times N_i \leq C. \tag{1}$$

We consider two different assumptions. The first assumes that jobs are like fluid and infinitely divisible. Any fraction of a job can be served. The second studies the situation when jobs are indivisible and non-preemptive. Every job has to be served in a whole unit, and the occupied resources will not be used by any other jobs during the execution of the job.

For the first case, we can define an instantaneous fairness at every time point. Therefore, we do not need the information on when the workflows arrive and depart. Instead, we assume that at a generic time point, all workflows are processed simultaneously in the shared cluster. Since N_{ij} can take any positive real values, in view of (1), it is clear that the set $N_i^S = \{N_i \in R^J : R_i \times N_i \leq C, N_i \geq 0\}$ forms a convex set. For each workflow i, we can characterize a Pareto curve for the vector N_i by assuming that the whole cluster with resources C is dedicated to service i. Specifically, let N_i^P be the maximal subset of N_i^S such that

$$N_i^P = \left\{ z \in N_i^S : \text{there is no } w \in N_i^S \text{ such that } w > z \right\}, \tag{2}$$

which represents the set of optimal operation points when workflow i is served exclusively in the cluster. As multiple workflows are processed simultaneously on the same cluster, the operation point of each workflow will move below the optimal curve and lie within the set N_i^S.

To be fair for all workflows with divisible jobs, we can specify that the slowdowns of all workflows to be equal. However, it is possible that there are still leftovers for some types of resources while the other types are saturated, as already addressed in [17]. If a class of jobs does not need these saturated types of resources, then they can utilize the still available resources, which can further decrease their slowdowns. Therefore, eventually we will observe uneven slowdowns. This problem can be avoided if every job requires all types of resources. Otherwise, we need to run multiple rounds of job allocation. In each round, we only consider those types of classes that can still be increased using the available resources.

The second case is when jobs are non-preemptive and indivisible. Unlike the previous case that only needs to focus on a specific time point, we assume that workflow i arrives at time s_i and departs the cluster at time f_i. Every job, once started, will take some execution time, during which the occupied resources will not be used by others. We require that N_{ij} take only integer values. The set

$$N_i^S = \{N_i \in Z^J : R_i \times N_i \leq C, N_i \geq 0\} \tag{3}$$

contains discrete elements. We can still define the set of optimal operation points the same as in (2). Since N_{ij} only take integer values, it is possible that no further jobs can be launched while at the same time every types of resources are not saturated. For example, consider the available resources $(50, 100)$ with a job profile $(40, 80)$. Only one job can be launched and $(10, 20)$ will be the leftover resources. Though both types of resources have surpluses, they are not enough for serving a whole unit of job.

4. DIVISIBLE JOBS

In this section, we assume that the jobs are infinitely divisible $N_{ij} \in R^+$. Because of this assumption, we only need to investigate fairness at a generic time, since the following argument applies for every time point. As explained earlier, we need to introduce a definition of slowdown for mulit-class workflows. We say an allocation is fair for concurrent workflows if they experience the same slowdown. For an operation point $p \in N_i^S$ for workflow i, we define its slowdown η_i by relating p to a corresponding point $p^C \in N_i^P$.

Definition 1. The slowdown of workflow i at operation point $p \in N_i^S$ is defined as

$$\eta_i(p) = \inf \left\{ x : p/x \in N_i^S, x \in R^+ \right\}.$$

We can specify that all concurrent workflows have the same slowdown at any time, under the assumption that $R_{ij} > 0$

for all i, j. Without this assumption, the slowdowns do not need to be equal, as explained in Section 3.

There could be a constraint on the combinations of different classes of jobs that can run simultaneously for a workflow. For some applications, the percentages of different job classes of a workflow remain fixed, i.e., the ratios between N_{ij} being fixed for different j. In this situation, multiple jobs of different classes are combined proportionally, which essentially is equivalent to a single class workflow. For other scenarios, a workflow has the flexibility to change the percentages of running jobs of different classes. Depending on how different jobs are mixed together, the workflow slowdown can also change accordingly. Therefore, a decision on the optimal combination of different classes of jobs is critical. Our objective is to maximize the slowdowns of all concurrent workflows. In addition, we want to fully utilize all available resources and maximize the resource usages.

In this section, we consider a special case with only two types of resources r_1, r_2, e.g., CPU and memory. We normalize the demand profile such that the largest element is equal to 1. Because of this normalization, the demand profiles are on the set $\{(r_1, r_2) : 0 \leq r_1 \leq 1, r_2 = 1\} \cup \{(r_1, r_2) : 0 \leq r_2 \leq 1, r_1 = 1\}$. Denote by $P_1^i, P_2^i, \cdots, P_J^i$ the profiles of workflow i with $P_j^i = (p_{j1}^i, p_{j2}^i)$, and by $x_1^i, x_2^i, \cdots, x_J^i$ the percentages for mixing these job profiles together, $\sum_{j=1}^J x_j^i = 1$. Then, the equivalent profile of mixing these jobs is equal to

$$P^i = (p_1^i, p_2^i) = \frac{\sum_{j=1}^J x_j^i P_j^i}{||\sum_{j=1}^J x_j^i P_j^i||_\infty}, \tag{4}$$

where $||(y_1, y_2)||_\infty = \max(y_1, y_2)$.

For workflow i, suppose that $N_{ij}/\sum_j N_{ij}$ has a constraint that it is within an interval $[\xi_{il}, \xi_{ir}]$. Then, the feasible combined profiles can be shown to lie in a close set. Using the mapping defined by (4), we can compute the set S_i of normalized profiles for workflow i.

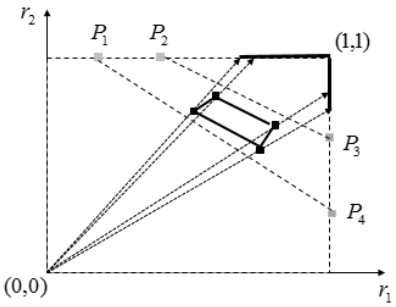

Figure 3: Equivalent profile for mixing 4 profiles

First, we maximize the slowdown by the following procedure. In the set S_i, we can always find out an edge subset S_i' containing at least one but at most two points in the form of either $(1, y_i)$ or $(x_i, 1)$, where x_i, y_i are minimal. We find the solution to the following optimization

$$\zeta = \min_{p_j^i \in S_i'} \max_{j=1,2} \sum_i p_j^i, \tag{5}$$

and the maximal slowdown η is equal to $1/\zeta$. This solution also defines a corresponding operation point, at which at least one of the resources is saturated. If both resources are saturated, it gives the optimal operation point. Otherwise, there is a type of resource r^{ast} that still has leftover. The next step is to increase the resource usage of r^{ast}. To this end, we adjust the mixing of the classes of jobs. For all jobs with profiles that do not have a dominant share for r^{ast}, we can increase the value of the profile for r^{ast} until either r^{ast} is saturated or this value cannot be increase anymore. In this step, it is possible to obtain multiple optimal operation points. Thus, the result is not always unique.

Using the aforementioned procedure, we can obtain the optimal operation points. It can be shown that these points all satisfy sharing-incentive in the sense that each job gets no less than a $1/n$ share of its dominant resource for n concurrent workflows. In addition, these allocations also satisfy envy-free property, which means that any workflow cannot get a better slowdown by using another workflow's allocation. An interesting result is that the envy-free property is not satisfied when the system works in a operation point that is fair but not efficient, i.e., all workflows having the same, but not maximal, slowdown. Furthermore, it can be shown that the system is strategy-proof that a workflow cannot increase its slowdown by modifying its edge subset S_i'. It is possible that a workflow has the same set S_i' after changing its job profiles.

5. INDIVISIBLE JOBS

The previous study assumes that jobs are infinitely divisible: if the allocated resources are not enough for an entire job, a fraction of the job can be launched. This is the assumption made in [8, 13, 14, 4], under which the fairness can be guaranteed instantaneously at each time.

However, for many applications, jobs are non-preemptive and each can only be launched as a whole unit. Therefore, instantaneous fairness is not always possible. For example, when a new workflow arrives and sees no available resources, it has to wait until some running jobs finish before it can launch some of its jobs. During the waiting period, the new workflow obtains no service at all, which is considered to be unfair if the fairness criteria only depends on the allocated resources at each time. Therefore, the instantaneous fairness cannot be used and an average slowdown over a time interval may be a better option.

In order to deal with this difficulty, we introduce an average fairness measured over a time interval, instead of the strict definition only depending on the allocated resources of a specific time point. The job profile of a workflow defines the required amount of resource of each type for every job of this user. Once a job is launched, it takes some time to execute. We assume that the job execution times of different classes are all upper bounded by a finite constant C. Since the jobs are non-preemptive, the occupied resources cannot be used by others during the execution. After the job finishes, it returns the resources to the cluster, which can be further allocated to other jobs. Similar to the divisible case, using the definition in (3), the slowdown of workflow i at operation point $p \in N_i^S$ is defined as

$$\eta_i(p) = \inf\left\{x : p/x \in N_i^S, x \in R^+\right\}.$$

In this section, we assume that each workflow has a single class, and thus we replace workflow by user to emphasize the difference. Suppose that user i arrives at a cluster at time s_i, and spend some time t_i in execution before it leaves the cluster at time f_i.

First, we use an example to illustrate that the instantaneous fairness that only relies on the allocated resources at the current time point can lead to unfair scheduling decisions when the jobs are non-preemptive and indivisible. Consider a cluster with 100 units of a single type of resource shared by user A and B. Both A and B has a lot of jobs to finish. Each job of user A requires 30 units of resources, and each job of user B requires 10 units of resources. In addition, we assume that a job of user A takes 100 units of time to finish, compared to a job of user B, which only needs 1 unit of time to finish. Suppose that at time 0 two jobs of user A and 4 jobs of user B run simultaneously in the cluster, which consumes $2 \times 30 + 4 \times 10 = 100$ resources. At time 100, the two jobs of user A finishes. Since user B only receives 40% of the resources, due to fairness, the scheduler will increase the number of running jobs for user B. Therefore, user A can only run one job at time 100. During the time period 100 to 101, user A runs one job and user B can run 7 jobs. When the jobs of user B finish at time 101, the scheduler notice that user A only receives 30% of the resources, and it will increase the number of running jobs for A to 2. Roughly speaking, the allocation of 2 jobs for user A and 4 jobs of user B running simultaneously will occur for most of the time during the course of execution. As the process continues, user A will get more computation resources from the cluster than user B over a long run. The percentages are approximately equal to 60% and 40% for user A and B, respectively.

This unfairness can be more pronounced when considering the fact that a cluster consists of many small computer nodes of different sizes. The resources allocated to jobs are from these separate discrete computer nodes, and a running job needs to use the resources from a single node. Therefore, if the resources on one computer node are not enough for a job, the job still cannot be served even though the total accumulated available resources on the whole cluster are abundant. We say that a set of resources are eligible for a user if they can be used to serve the respective jobs. Consider the following a cluster that has two subsets S_1 with 50 computer nodes of size $(10, 1000)$ and S_2 with 50 nodes of size $(4, 500)$. If the user A has the job profile $(5, 500)$, then the only eligible set of nodes are S_1 with 50 ones of size $(10, 1000)$. Though the user can use the whole cluster, it is as if the user has been deprived of the access to S_2. Therefore, when we compare fairness of user A with any other user, say B, we should not consider the usage of user B on S_2. Instead, we can consider the maximal set of resources that both A and B can use. They are consider to have fair shares if their allocation on that set of resources are fair.

Let $N_i(t) \in N$ denote the number of running jobs of user i at time t, with $N_i \in N$ being the maximum number of jobs that user i can be launched in the cluster assuming that no other users are present. The slowdown of user i at time t is equal to $\eta_i(t) = N_i(t)/N_i$. The average slowdown over a time interval $[a, b], b - a = l$ is defined as $\eta_i([a, b]) = \int_a^b \eta_i(t)dt/l$.

Definition 2. For two users, u_1 and u_2, who can also share resources with other users, an allocation is (l, ϵ)-fair for u_1 and u_2, if during any time interval $[a, b]$ of length $l = b - a$ when u_1 and u_2 are served simultaneously in the cluster, the ratio between the average slowdowns of u_1 and u_2 on $[a, b]$

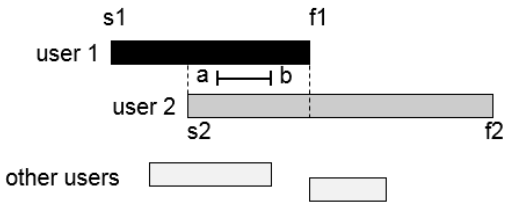

Figure 4: Illustration of fair allocation

are different by at most $1 + \epsilon, \epsilon > 0$, i.e.,

$$\frac{1}{1+\epsilon} \leq \frac{\eta_{u_1}([a,b])}{\eta_{u_2}([a,b])} \leq 1 + \epsilon.$$

Next, we propose a scheduling algorithm so that all sharing users are treated (l, ϵ)-fair, under the assumption that the execution times of all jobs are bounded by a constant C. We want to have the following property: for any given $\epsilon > 0$, there exists a finite $l > 0$ such that the scheduling algorithm guarantees all workflows to be (l, ϵ)-fair.

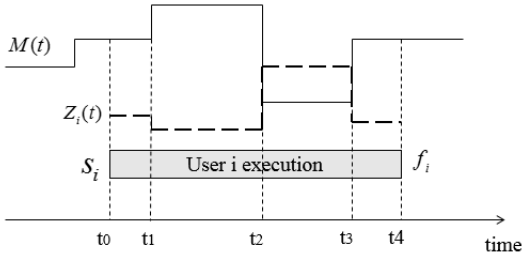

Figure 5: Dynamics of $M(t)$ and $Z(t)$

Let $M(t)$ be the number of users running simultaneously in the cluster at time t. The slowdown of user i is equal to $\eta_i(t)$. For a user that still has unfinished workload at time t, define

$$Z_i(t) = \frac{\eta_i(t)}{\sum_{n=1}^{M(t)} \eta_n(t)},$$

as illustrated in Fig. 5. The following function for user i at time $t > s_i$ on a time interval $[\max(t - l, s_i), t]$ is defined by

$$C_i(t) = \int_{\max(t-l,s_i)}^{t} Z_i(x)M(x)dx.$$

We can use $C_i(t)$ as a performance indication function, since ideally $C_i(t)$ should be very close to l for all users and any time t. Based on $C_i(t)$, a straightforward algorithm is to sort all users that are still running in the cluster according to $C_i(t)$ in ascending order at each time t, and launch a job from the user with the smallest $C_i(t)$ whenever possible.

This indicator function captures time dependence, which is missing in any approach that only specifies instantaneous fairness requirement. To see this point, consider an extreme case that each user can only run one job in the cluster. Without using the past information about the already accumulated service, we cannot fairly allocate the resources to multiple users. This becomes important when multiple users with different profiles are sharing the cluster together. For example, it is possible that some users have jobs requiring a lot of resources and other small jobs each only need a small

amount of resources. These small jobs can be squeezed into the resource "fragments" but large jobs cannot. We need to track the accumulated resource allocation over time.

When the jobs are infinitely divisible and the jobs have multi-dimensional profiles, it can be shown that the previous algorithm is equivalent to dominant resource fairness as $l \to 0$. Furthermore, if all job profiles are identical, this algorithm is the same as processor sharing when $l \to 0$.

In the multi-resource scenarios, carefully packing jobs of various profiles can help to fully utilize the resources of different types. However, if the fairness criteria is too strict, there will not be enough room for the scheduling algorithm to optimize the packing of jobs.

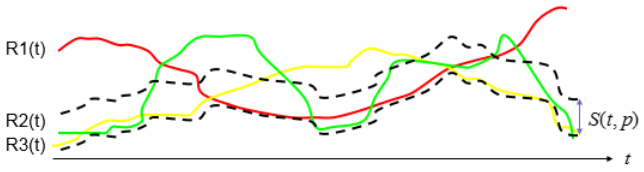

Figure 6: Relax the candidates for launching jobs

In order to make room for further optimization, we change the algorithm to the following. Still, we sort all users that are still running in the cluster according to $C_i(t)$ in ascending order at each time t. For the user u^* with the smallest $C_{u^*}(t)$, we define a set of users $S(t,p) = \{i : C_i(t) \leq (1+p)C_{u^*}(t)\}$ for a fixed $p > 0$. Every user in the set $S(t,p)$ is eligible to launch a job immediately after t. The decision on which user to be selected can be determined by other objective functions, e.g., improving resource utilization by bin-packing algorithms [9].

The reason why this algorithm maintains fairness can be heuristically argued as follows. If a user v always obtains less than its desired fraction of resources, its $C_v(t)$ value will keep decreasing. This process eventually leads the set $S(t,p)$ to only contain the single user v. It thus will be allocated with resources to catch up with other users that may have temporarily been treated more advantageously.

6. RELATED WORK

As the first attempt of solving the multi-resource allocation problem for distributed computing, Ghodsi et al. [8] propose a dominant resource fair sharing (DRF) mechanism for multi-resource allocation in large scale cloud computing clusters. DRF generalizes max-min fairness sharing of single resource type to multiple resource type with a goal of maximizing the minimum dominant share for all users. It also satisfies a number of useful game theory properties such as sharing incentive (SI), Pareto efficiency (PE), strategy proofness (SP) and envy-freeness (EF). DRF assumes that the resource pools are homogeneous and infinitely divisible which is essentially equivalent to provide a single gigantic machine to meet the resource requests from all users.

After DRF, multi-resource allocation has gained significant attention [17, 5, 10, 13, 14] from both computer science and economics communities. Dolev et al. [5] introduce a new way of defining fairness based on bottleneck resources. They propose bottleneck-based fairness (BBF) allocation methods in which users are assigned at least the entitlement of their bottlenecked resources and cannot be justified for complaints. They then theoretically prove the existence of fair allocations that satisfy no justified complaints. Later on, a number of techniques generalize DRF to either consider a larger classes of fairness functions or to account for more general cases [10, 13, 17]. Gutman and Nisan [10] extend and generalize BBF [5] and DRF [8] to a larger family of utilities under Leontief preference from an economic perspective. A new class of fairness notion, named generalized resource fairness (GRF), are defined based on which the authors design two polynomial time algorithms for fair allocation. Note that the second polynomial algorithm targets an open problem of computing a BBF allocation [5] relying on a corollary of finding competitive market equilibria in a Fisher market. Joe-Wong et al. [13] generalize DRF and present two classes of fairness functions including fairness on dominant shares (FDS) and generalized fairness on jobs (GFJ). Through these functions, they explore the trade-off between efficiency and fairness, study the impact of user requests with multi-resource types and discuss conditions where these fairness functions satisfy the game theory properties. In these techniques, they all assume resources are continuously divisible. Parkes et al. [17] present another research effort that generalizes the DRF definition to account for per job sharing weights and zero demands, and that compares with related work in economics community using Leontief preference. They also investigate the social welfare properties of fair allocation algorithms. Given the assumption that user requests are indivisible, they design a polynomial sequential min-max algorithm which allows job arrival and departure as well as changing task profiles over time. They discuss the possibility and impossibility of fairness properties when requests are indivisible and provide results with relaxed notion of fairness under a certain setting. Our problem setting is different from the above literature in that we consider multiple different task profiles of a single user request instead of the assumption that each user request is associated with a task profile.

Two independent approaches [4, 14] investigate the impact of dynamic job arrival and departure on fairness and efficiency of multi-resource allocation problem. Inspired by the proportional fairness of bandwidth sharing in networks, Bonald et al. [4] argue that proportional fairness can obtain better efficiency -fairness trade-off. Kash et al. [14] on the other hand introduce new notion of dynamic Pareto optimality (DPO) and show that DPO and envy-free are incompatible. To tackle the incompatibility, they propose two relaxations: 1) dynamic envy-free (DEF) based on which a dynamic DRF allocation method is proposed to satisfy SI, DEF, SP and DPO; 2) cautious DPO based on which cautious LP are designed to satisfy SI,EF, SP and CDPO. Both models assume divisible jobs.

While a large portion of related prior art has concentrated on fair allocation under the assumption that both the resource requests of jobs are divisible, Friedman et al. [6] study the multi-resource fair sharing under a more realistic assumption that resource requests of jobs are indivisible and there are a number of machines which provide available resources. Their approach also considers the environments where containers are used to achieve performance isolation between tenants. Under this setting, the authors show that a randomized resource allocation with approximate ex-post fair, efficient and strategy-proof can be achieved by computing a weighted max-min over the convex hull of the feasible

region. Another research work by Psomas and Schwartz [19] considers the cases that task resources are indivisible and applies bin packing algorithm for allocation. They demonstrate that the designed mergerDRF algorithm can increase utilization.

In addition, DRF has been extended and applied in other fields such as network routing [7, 20], cluster resource allocation for hierarchical organizations [3] and IaaS cloud [16]. Egalitarian division [15] under Leontief preferences and the cake-cutting problem [18] discuss fair resource allocation problem in the economic context with an assumption that resources are divisible. These efforts are different from our work in that we handle the problem in a more realistic setting where a single user submits multiple different task profiles concurrently.

7. CONCLUSION

We generalize the notion of slowdown to address the fairness for multi-class workflows that demand multiple different resources. Two cases are considered when jobs are divisible and indivisible, respectively. For the first case when the operation point of a workflow moves dynamically as the scheduling decisions change, we compare the instantaneous operation point with one that is mapped onto a Pareto curve by assuming that the workflow has the whole cluster at its disposal. We investigate fairness among concurrent workflows using slowdown and optimize the mixture of different classes of jobs for a workflow for better slowdown. Then, for the second case, in order to handle the situation when jobs are non-preemptive and indivisible, we relax the strict fairness definition that only depends on the consumed resources at a particular time to an average slowdown measured over a time interval. This relaxation allows us to address fairness for more realistic application scenarios.

References

[1] Hadoop. http://hadoop.apache.org/.

[2] Spark. http://spark.apache.org/.

[3] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. 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.

[4] T. Bonald and J. Roberts. Enhanced cluster computing performance through proportional fairness. pages 2–23, Turin, Italy, October 2014.

[5] D. Dolev, D. G. Feitelson, J. Y. Halpern, R. Kupferman, and N. 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.

[6] E. Friedman, A. Ghodsi, and C.-A. Psomas. Strategyproof allocation of discrete jobs on multiple machines. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, EC '14, pages 529–546, New York, NY, USA, 2014. ACM.

[7] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica. Multi-resource fair queueing for packet processing. *SIGCOMM Comput. Commun. Rev.*, 42(4):1–12, Aug. 2012.

[8] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. 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 323–336, Berkeley, CA, USA, 2011. USENIX Association.

[9] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 455–466, New York, NY, USA, 2014. ACM.

[10] A. Gutman and N. Nisan. Fair allocation without trade. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 719–728. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[11] M. Harchol-balter, K. Sigman, and A. Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. October 2002.

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[13] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *INFOCOM, 2012 Proceedings IEEE*, pages 1206–1214, March 2012.

[14] I. Kash, A. D. Procaccia, and N. Shah. No agent left behind: Dynamic fair division of multiple resources. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 351–358, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

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

[16] H. Liu and B. He. Reciprocal resource fairness: Towards cooperative multiple resource fair sharing in iaas clouds. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 970–981. IEEE Press, 2014.

[17] D. C. Parkes, A. D. Procaccia, and N. 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.

[18] A. D. Procaccia. Cake cutting: Not just child's play. *Commun. ACM*, 56(7):78–87, July 2013.

[19] C.-A. Psomas and J. Schwartz. Beyond beyond dominant resource fairness: Indivisible resource allocation in clusters. Technical report, Tech Report Berkeley, 2013.

[20] W. Wang, B. Li, and B. Liang. Multi-resource round robin: A low complexity packet scheduler with dominant resource fairness. In *ICNP*, pages 1–10, 2013.

[21] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an m/gi/1. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, pages 238–249, New York, NY, USA, 2003. ACM.