QKnober: A Knob-based Fairness-Efficiency Scheduler for Cloud Computing with QoS Guarantees

Shanjiang Tang[1*], Ce Yu[1*], Chao Sun[1], Jian Xiao[1], Yinglong Li[2]

School of Computer Science & Technology, Tianjin University[1]
School of Computer Science & Technology, Zhejiang University of Technology[2]
{tashj,yuce,sch,xiaojian}@tju.edu.cn, liyinglong@ruc.edu.cn

Abstract. Fairness and efficiency are generally two important metrics for users in modern cloud computing. Due to the heterogeneous resource demands of CPU and memory for users' tasks, it cannot achieve the strict 100% fairness and the maximum efficiency at the same time. Quantitatively showing the fairness degradation/loss becomes essentially important in the design of any fairness-efficiency tradeoff scheduler. Existing fairness-efficiency schedulers (e.g., Tetris) can balance such a tradeoff elastically by relaxing fairness constraint for improved efficiency using the knob. However, their approaches are *insensitive* to the fairness degradation under different knobs, which makes several drawbacks. First, it cannot quantitatively tell how much *relaxed* fairness can be guaranteed (i.e., QoS of fairness guarantee) given a knob value. Second, it fails to meet several essential properties such as sharing incentive. To address these issues, we propose a new fairness-efficiency scheduler, *QKnober*, to balance the fairness and efficiency elastically and flexibly using a tunable fairness knob. QKnober is a *fairness-sensitive* scheduler that can maximize the system efficiency while guaranteeing the θ-soft fairness by modeling the whole allocation as a combination of *fairness-purpose* allocation and *efficiency-purpose* allocation. Moreover, QKnober satisfies fairness properties of sharing incentive, envy-freeness and pareto efficiency given a proper knob. We have implemented QKnober in YARN and evaluated it using real experiments. The results show that QKnober can achieve good performance and fairness.

1 Introduction

In the current era of 'big data', it has become typical to take existing large-scale data computing frameworks such as MapReduce [8] and Spark [26] for big data analytics in a cloud system consisting of many machines [16]. At any time, there are many users running their data-parallel applications on the cloud. Typically, users' submitted jobs often contain many tasks and their tasks tend to have *heterogeneous* resource requirements towards different resource types (e.g., CPU and memory). For example, tasks of machine learning applications are CPU-intensive [10], whereas hash join and sort tasks of database queries are memory-intensive [6].

Fairness and efficiency are generally two critical metrics for both users and resource providers in cloud computing [12]. Being aware of heterogeneous resource demands of users' tasks, there is a need to consider multi-resource fairness that takes multiple resource types into account. By leveraging the game-theoretic definition, a *robust* multi-resource fair allocation is the one in which

– *all users in the shared system should perform no worse than that under an exclusively non-sharing partition of the system. (Sharing incentive)*
– *no user envies the allocations of any other users. (Envy freeness)*
– *no user can increase its resource allocation with harming at least one other user. (Pareto efficiency)*

Dominant Resource Fairness (DRF) is one of the most well-known multi-resource fair allocation policies [9] with the above three game-theoretic properties. It introduces the concept of dominant resource, referred to as the resource that is heavily used by a user. The fairness is achieved by equalizing the share of each user's dominant resource. Although there have since been a number of extensions [13,23], they draw little attention to the influence on system efficiency. Recent studies have shown that there is a tradeoff between fairness and efficiency in multi-resource allocation [10,11,22]. Guaranteeing the strict 100% fairness across users would produce inefficient resource allocations. Conversely, seeking for high system efficiency is often at the cost of compromised fairness. DRF and its extensions tend to over constrain the system for high fairness guarantee, resulting in resource allocations with low system efficiency.

Many existing fairness-efficiency schedulers seek to relax fairness (i.e., allowing some degree of unfairness) for efficiency improvement by employing knob-based heuristic algorithms [7,10,17,24]. Quantifying the fairness degradation/loss is essentially important for users in order to properly configure the knob value. Tetris [10] is the state-of-the-art knob-based tradeoff scheduler that allows users to balance fairness and efficiency flexibly by tuning the fairness knob in cloud computing. However, due to its *insensitiveness* of fairness degradation under different knobs, there are some shortcomings (See Section 3): 1). it cannot quantitatively show users how much *relaxed* fairness can be guaranteed given a fairness knob (i.e., QoS of fairness guarantee); 2). it fails to satisfy several fairness properties such as sharing incentive.

In this paper, we develop a new fairness-efficiency scheduler, *QKnober*, to allow users to balance fairness and efficiency flexibly with a knob factor $\rho \in [0, 1]$. Unlike the previous schedulers [7,10,24], QKnober is a *fairness-sensitive* scheduler that works on the *relaxed* fairness (i.e., *soft* fairness in Section 4.1), which refers to the maximum difference between the normalized shares of any two users. It is achieved by modeling the multi-resource allocation as a combination of *fairness-purpose* allocation and *efficiency-purpose* allocation (Section 4.1). Given a knob ρ, QKnober first performs the fairness-purpose allocation for the QoS of θ-soft fairness guarantee (See Theorem 1 in Section 4.1) and then does the efficiency-purpose allocation for maximizing the system efficiency. We show that with a proper knob configuration, QKnober can ensure that each user in the shared system can get at least the amount of resources as that under the exclusively non-sharing partition of the system. It also can guarantee that every user prefers to its own allocation and no user envies the allocations of any other users. Furthermore, QKnober keeps that the system is fully utilized by ensuring that no user can get more resource allocation without decreasing the allocation of at least one user.

We have implemented QKnober in YARN [20]. We evaluated QKnober with testbed workloads in a Amazon EC2 cluster consisting of 60 nodes. Our results show that QKnober strikes a flexible balance between fairness and efficiency. There can be up to 57% performance improvement as we decrease the knob factor from one to zero for

QKnober. Moreover, it outperforms its alternatives DRF and Tetris by 31.2% and 4.5% on average, respectively. Finally, we show that the scheduling overhead of QKnober is minor (< 0.42 ms).

2 Desirable Allocation Properties

From the economic point of view, a *good* fair allocation policy in cloud computing system should provide the following essential game theoretic properties, including sharing incentive, envy-freeness, and pareto efficiency [9].

Sharing Incentive (SI): Resource sharing is an essential and effective approach to improve the system utilization and efficiency [15]. A good allocation policy should satisfy sharing incentive (SI) such that each user in the system performs at least as good as it would be under a statically equal split of the resources of the computing system. Otherwise, users would be more likely to divide the computing system equally and exclusively use their own partitions without sharing. Thus, to enable resource sharing possible and sustainable, it is a must requirement to satisfy sharing incentive [18].

Formally, let $\mathbf{U}_i = \langle u_{i,1}, \cdots, u_{i,m} \rangle$ be the resource allocation vector for user i. Let $N_i(\mathbf{U}_i)$ denote the number of tasks scheduled for user i under the resource allocation vector $\mathbf{U}_i$. An allocation policy is sharing incentive if it satisfies the following condition for each user $i \in [1, n]$,

$$N_i(\mathbf{U}_i) \geqslant N_i(\overline{\mathbf{U}}_i), \tag{1}$$

where $\overline{\mathbf{U}}_i = \langle \overline{u}_{i,1}, \cdots, \overline{u}_{i,m} \rangle$ represents the resource allocation vector for user i under the exclusively non-sharing partition of the computing system.

Envy-freeness (EF): An allocation is envy-freeness (EF) if no user envies the allocation of other users associated with a desire to receive that same allocation. That is, every user prefers its own allocation to that of any other user. To provide EF, there is a need to ensure that every user cannot have more tasks scheduled by switching its allocation with any other user.

Given the resource allocation vector $\mathbf{U}_i$ for user i, an allocation policy satisfies EF if

$$N_i(\mathbf{U}_i) \geqslant N_i(\mathbf{U}_j), \tag{2}$$

for any two users $i, j \in [1, n]$.

Pareto Efficiency (PE): PE is another critical property that should be satisfied by a fair resource allocation policy [19]. It is essential for high resource utilization and efficiency. An allocation policy is PE if it is not possible for a user to get more tasks scheduled without decreasing the number of running tasks of at least one other user.

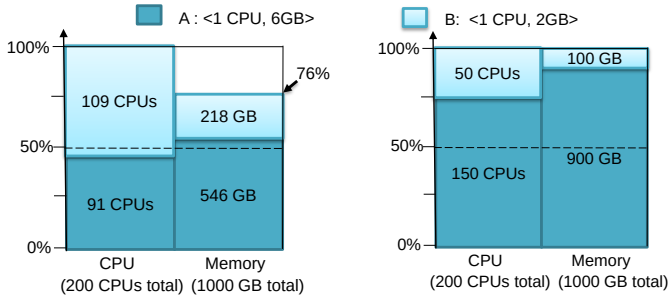
Let $\mathbf{U} = \langle \mathbf{U}_1, \cdots, \mathbf{U}_n \rangle$ be the resulting allocation for all users produced by a fair allocation policy. The allocation $\mathbf{U}$ is PE if it does not exist any feasible allocation $\breve{\mathbf{U}}$ satisfying the following two conditions at the same time, i.e., 1). $\forall i \in [1, n], N_i(\mathbf{U}_i) \leqslant N_i(\breve{\mathbf{U}}_i)$; 2). $\exists j \in [1, n], N_j(\mathbf{U}_j) < N_j(\breve{\mathbf{U}}_j)$.

3 Background and Motivation

In this section, we motivate our work by reviewing and analyzing the limitations of existing schedulers.

***Fairness* vs. *System Efficiency*.** In multi-resource allocation, the fairness and system utilization/efficiency highly depend on the workload characteristic and allocation ratio of the resource. If the users with memory-intensive workloads have small allocation ratio, it may result in low utilization for the memory resource due to insufficient requests. Similar case does also hold for other resources (e.g., CPU). On the contrary, maintaining high resource utilization for all resources often generate the allocations in a manner that starves some users, resulting in unfairness problem for users in the allocation. We next demonstrate these problems using examples of Dominant Resource Fairness (DRF) [9], which is a popular multi-resource allocation policy with many attractive merits (e.g., sharing incentive, envy-freeness and pareto-efficiency).

Example 1. Consider a computing system consisting of 200 CPUs and 1000 GB memory in total. It is shared by two users A and B equally with the task requirement of $\langle$1 CPU, 6 GB$\rangle$ for A and $\langle$1 CPU, 2 GB$\rangle$ for B, respectively.



(a) DRF allocation. (b) Pure efficiency allocation.

Fig. 1: Allocation results with different policies for Example 1. The memory utilization is only 76% for DRF, whereas we can achieve 100% utilization for CPU and memory with pure efficiency policy.

The dominant resource for user A is memory because each task of A consumes $1/200$ of the total CPUs and $6/1000$ of the total memory, while the dominant resource for B is CPU. DRF achieves the fairness by equalizing the dominant resource shares (i.e., $546/1000 = 109/200$) for A and B, with the resulting allocation illustrated in Figure 1 (a). The memory utilization is only $\frac{(546+218)}{1000} \approx 76\%$. This is because DRF does not consider resource efficiency when making allocation decision. It only focuses on achieving the fairness among users, but does not deal with the impact of such adjustments on the system efficiency.

In fact, both CPU and memory resources in Example 1 can be fully utilized if the scheduler allocates $\langle 150$ CPUs, 900 GB$\rangle$ to A and $\langle 50$ CPUs, 100 GB$\rangle$ to B, as illustrated in Figure 1 (b). However, the dominant resource shares of A and B are no longer the same (i.e., $\frac{900}{1000} > \frac{50}{200}$), being *unfair* for B. It implies that there tends to be a tradeoff between fairness and system efficiency in resource allocation.

Flaws of the *State-of-the-Art* Tradeoff Scheduler for Cloud Computing. To balance the tradeoff between fairness and efficiency elastically and flexibly, many fairness-efficiency schedulers [10,17,22,24] take *knob-based* heuristics, which is *promised* as an effective approach in multi-resource allocation [10]. Wang et al. [22,24] studied the fairness-efficiency tradeoff in networking system by considering network packet processing and data transfer flow across different machines. EMRF [17] is a fairness-efficiency tradeoff scheduler for Coupled CPU-GPU architectures. In contrast, for cloud computing, Tetris [10] is the *state-of-the-art* knob-based scheduler. Specifically, in each resource allocation, it first sorts all tasks according to the DRF. Then, it searches the best task for efficiency among the runnable tasks belonging to the first $(1-f)$ tasks in the sorted list, where $f \in [0,1]$ is a knob provided by users in advance. It computes the *alignment score*, defined as the weighted dot product between the vector of machine's available resources and the task's peak resource demand, to the machine for each task, and the best task is picked with the largest alignment score. However, there are several flaws for Tetris as follows:

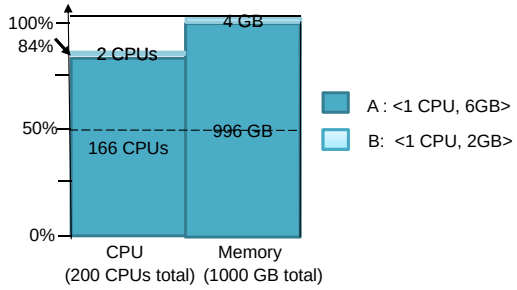


Fig. 2: The resulting allocation with Tetris for Example 1 when the knob f satisfies $0 \leqslant f < 0.5$. In this case, different value of knobs does not work for fairness improvement, indicating that Tetris is *insensitive* to fairness under different knobs.

First, although Tetris allows users to relax fairness for efficiency improvement by tuning the fairness knob, it is *insensitive* to fairness degradation for different knobs during the allocation. Particularly, it cannot quantitatively show users how much *relaxed* fairness (i.e., *soft* fairness in Section 4.1) can be guaranteed (i.e., QoS of fairness guarantee) given a knob configuration. To explain it, let's revisit Example 1 by assuming that at each allocation, there are 500 tasks for A and B, respectively. We can see that the task (share: $\langle \frac{1}{200}, \frac{6}{1000} \rangle$) of A is more beneficial to the system utilization than that (share: $\langle \frac{1}{200}, \frac{2}{1000} \rangle$) of B according to the resource type difference of their tasks ($A : |\frac{1}{200} - \frac{6}{1000}| = \frac{1}{1000}$, $B : |\frac{1}{200} - \frac{2}{1000}| = \frac{3}{1000}$). At each allocation, Tetris first sorts all 1000 tasks of A and B according to DRF policy. Then, it tries to pick up a task from the first $(1-f)$ tasks in the sorted list that is most beneficial to the system utilization. When $0 \leqslant f < 0.5$, the task range that Tetris can choose at the second stage is $500 < (1-f) * 1000 \leqslant 1000$. In this case, Tetris always picks up preferred tasks from A rather than B until it cannot fulfilled, resulting in allocation as shown in Figure 2. It shows that the knob of Tetris does not work for fairness improvement when $0 \leqslant f < 0.5$, i.e., Tetris is *insensitive* to fairness degradation under different knobs,

implying that it cannot tell users how much relaxed fairness can be guaranteed under a knob setting. However, in practice, the QoS guarantee of different levels of fairness is very important for users under different knobs configurations.

Second, Tetris violates the sharing incentive property (See definition in Section 2). Let's take Example 1 as a counterexample to demonstrate it. Provided that $f = 0$, Tetris is purely for system efficiency optimization by picking the best task for efficiency among all tasks every time, resulting in the allocation as illustrated in Figure 1 (b). We can see that, B receives less resources (i.e., fewer tasks scheduled) in the sharing system than that (i.e., $\langle 100$ CPUs, 200 GB$\rangle$) of exclusively using its partition of the system without sharing, violating the sharing incentive property.

Motivated by these, we seek to explore a new fairness-efficiency allocation policy that guarantees the soft fairness and satisfies all the desirable properties listed in Section 2.

4 Allocation Model and Scheduling Policy

In this section, we model the multi-resource allocation in cloud computing based on DRF, and propose a proposed fairness-efficiency scheduling policy called QKnober.

4.1 Multi-Resource Allocation Model

1) Basic Setting. We start by defining some terms used in our model. Suppose that the computing system consists of m resource types (e.g., CPU, memory, disk) with the capacity of $\mathbf{R} = \langle r_1, \cdots, r_m \rangle$ shared by n users, where r_i denotes the total amount of resource i. For each user i, let w_i denote its share weight in the shared computing system and $\mathbf{D}_i = \langle d_{i,1}, \cdots, d_{i,m} \rangle$ be its *resource demand vector*, where $d_{i,j}$ denotes the amount of resource j required by a task of user i. We assume that each user has an infinite number of tasks to be scheduled, and all its tasks are divisible and with the same resource demand. We later discuss how these assumptions can be relaxed for practical usage in Section 5.

Given the allocation matrix $\mathbf{U} = \langle \mathbf{U}_1, \cdots, \mathbf{U}_n \rangle$ for all users, it is a *feasible* allocation if it satisfies that,

$$\sum_{i=1}^{n} u_{i,k} \leqslant r_k, \quad \forall k \in [1, m], \tag{3}$$

The maximum number of tasks $N_i(\mathbf{U}_i)$ (possible fractional) that user i can schedule under the resource allocation vector $\mathbf{U}_i$ is,

$$N_i(\mathbf{U}_i) = \min_{1 \leqslant k \leqslant m} \{u_{i,k}/d_{i,k}\}, \tag{4}$$

2) Allocation Model. An efficient resource allocation should never let a user get more resources than it actually needs in the computing system. We call such an allocation *non-wasteful*. Formally, an allocation $\mathbf{U}_i$ is *non-wasteful* if and only if it satisfies the following condition:

$$\mathbf{U}_i = N_i(\mathbf{U}_i) \cdot \mathbf{D}_i, \tag{5}$$

It is worthy mentioning that we can always convert an allocation to the *non-wasteful* allocation by transferring the redundant/unused resources of each user to other potential users without decreasing the number of tasks scheduled for that user. Without loss of generality, in the following discussions, we limit our focus on the non-wasteful allocation.

Scheduling tasks to the computing system is analogous to the multi-dimensional knapsack problem [5] by viewing the computing system as a knapsack and each task as a knapsack item. The weight of an item (or task) from user i is $\mathbf{D}_i$. In this work, since we are interested in the efficiency of resource allocation, the value of an item (or task) is the sum of the amount of different typed resources it required (normalized to the system capacity), i.e., $\sum_{k=1}^{m} d_{i,k}/r_k$. Let $\epsilon_i(\mathbf{U}_i)$ be the efficiency (i.e, knapsack cost value) of a feasible resource allocation $\mathbf{U}_i$ contributed by user i in the system. According to the knapsack problem, we have

$$\epsilon_i(\mathbf{U}_i) = N_i(\mathbf{U}_i) \cdot \sum_{k=1}^{m} d_{i,k}/r_k, \tag{6}$$

for a single user i. Then the efficiency $\epsilon(\mathbf{U})$ of a feasible allocation $\mathbf{U}$ for all users in the system can be calculated as

$$\epsilon(\mathbf{U}) = \sum_{i=1}^{n} \epsilon_i(\mathbf{U}_i) = \sum_{i=1}^{n} \{N_i(\mathbf{U}_i) \cdot \sum_{k=1}^{m} d_{i,k}/r_k\}. \tag{7}$$

Let s_i denote the share of dominant resource for user i in the computing system. According to Formula (5), we have

$$s_i = \max_{1 \leqslant k \leqslant m} u_{i,k}/r_k = N_i(\mathbf{U}_i) \cdot \max_{1 \leqslant k \leqslant m} d_{i,k}/r_k. \tag{8}$$

Formula (8) indicates that there is a proportional relationship between a user's dominant resource share and the number of tasks scheduled. The Dominant Resource Fairness (DRF) achieves the fairness by guaranteeing that the (weighted) shares of dominant resource across users are the same, i.e.,

$$\frac{s_1}{w_1} = \frac{s_2}{w_2} = \cdots = \frac{s_n}{w_n}. \tag{9}$$

Let s_i^{max} and $N_i(\mathbf{U}_i^{max})$ represent the maximum share of dominant resource and the corresponding number of tasks scheduled for user i under the DRF allocation. The DRF allocation can be viewed as progressive filling when all tasks are divisible [9]. The allocation terminates when at least one typed resource is fulfilled. In that case, we are unable to increase each user's dominant resource. That is, the dominant resource share and the corresponding number of tasks scheduled are maximized for each user under DRF. It thus holds,

$$\max_{1 \leqslant k \leqslant m} \{\sum_{i=1}^{n} \frac{u_{i,k}}{r_k}\} = \max_{1 \leqslant k \leqslant m} \{\sum_{i=1}^{n} \frac{N_i(\mathbf{U}_i) \cdot d_{i,k}}{r_k}\} = 1. \tag{10}$$

By computing $N_i(\mathbf{U}_i)$ with Formula (8) (9) (10), we can derive $N_i(\mathbf{U}_i^{max})$ as follows:

$$N_i(\mathbf{U}_i^{max}) = w_i / (\max_{1 \leqslant k \leqslant m} \{\frac{d_{i,k}}{r_k}\} \cdot \max_{1 \leqslant k \leqslant m} \{\frac{1}{r_k} \cdot \sum_{j=1}^{n} \frac{w_j \cdot d_{j,k}}{\max_{1 \leqslant k' \leqslant m} \{\frac{d_{j,k'}}{r_{k'}}\}}\}).$$

According to Formula (8), we can get s_i^{max} as

$$s_i^{max} = w_i / \max_{1 \leqslant k \leqslant m} \{\frac{1}{r_k} \cdot \sum_{j=1}^{n} \frac{w_j \cdot d_{j,k}}{\max_{1 \leqslant k' \leqslant m} \{\frac{d_{j,k'}}{r_{k'}}\}}\}. \tag{11}$$

4.2 QKnober

Recall that the model in Section 4.1 is a strict 100% fairness allocation model. By altering the model slightly, we can develop a knob-based fairness-efficiency scheduler, *QKnober*, to allows users to balance fairness and system efficiency flexibly using a fairness knob.

The basic idea is as follows. Instead of strictly seeking for 100% fairness as DRF does, we can compromise fairness for increased allocation efficiency by tolerating some degree of fairness loss. Particularly, we classify the fairness into two types: *hard fairness* and *soft fairness*. The hard fairness refers to that the allocation shares of all users should be the same (i.e., Formula (9) should be guaranteed). In contrast, the soft fairness tolerates some degree (measured by θ) of unfairness across users. Formally, we define θ-*soft fairness* by changing Formula (9) as follows:

$$|\frac{s_i}{w_i} - \frac{s_j}{w_j}| \leqslant \theta, \forall i, j \in [1, n]. \tag{12}$$

Typically, DRF focuses on the hard fairness across users, limiting the allocation efficiency improvement. In contrast, QKnober, as a fairness-efficiency tradeoff scheduling policy, is interested in the soft fairness, which can leave some room for efficiency improvement. In the following, we describe our design of QKnober policy.

1) QKnober Design. The fairness-efficiency tradeoff allocation can be considered as an integration of two stages allocations: *fairness-purpose* allocation (i.e., purely for fairness optimization) and *efficiency-purpose* allocation (i.e., purely for efficiency optimization). For QKnober, it first does the fairness-purpose allocation with DRF to achieve the soft fairness guarantee. Next it turns to the efficiency-purpose allocation for efficiency maximization. Particularly, it offers users a knob $\rho \in [0, 1]$ to control and balance the two stages allocations flexibly. Let $\bar{s_i}$ and $s_i^{'}$ be the dominant resource shares of the resulting allocation for user i in the stage of fairness-purpose allocation and efficiency-purpose allocation, respectively. By combining the allocations of two stages, we get the final dominant resource share s_i for each user i as follows:

$$s_i = \bar{s_i} + s_i^{'}. \tag{13}$$

Fairness-purpose Allocation. In the stage of fairness-purpose allocation, instead of guaranteeing the hard (dominant resource) fairness of s_i^{max} for each user i, QKnober seeks to guarantee the soft fairness of $s_i^{max} \cdot \rho$ (i.e., $\bar{s_i} = s_i^{max} \cdot \rho$). According to Formula (13), we have

$$s_i = s_i^{max} \cdot \rho + s_i^{'}. \tag{14}$$

It leaves $\mathbf{R}^{'} = \langle r_1^{'}, \cdots, r_m^{'} \rangle$ idle resources for efficiency-purpose allocation, where $r_i^{'} = r_i - \sum_{j=1}^{n} \frac{s_j^{max} \cdot \rho \cdot d_{j,i}}{\max_{1 \leqslant k \leqslant m} d_{j,k}/r_k}$ according to Formula (8). The small value of ρ favors the efficiency optimization. In contrast, the large value of ρ can make the fairness-purpose allocation dominant, benefiting more for fairness optimization. Typically, QKnober reduces to DRF when $\rho = 1$.

Theorem 1. *QKnober is a θ-soft fairness policy where*

$$\theta = \max_{1 \leqslant i \leqslant n} \{ \frac{\max_{1 \leqslant k \leqslant n} d_{i,k}/r_k}{w_i \cdot \max_{1 \leqslant k \leqslant m} \{ \frac{d_{i,k}}{r_k - \sum_{j=1}^{n} \frac{s_j^{max} \cdot \rho \cdot d_{j,k}}{\max_{1 \leqslant k' \leqslant m} d_{j,k'}/r_{k'}}} \}} \}.$$

The proof of Theorem 1 can be found in Appendix A of Technique Report [14].

Efficiency-purpose Allocation. Theorem 1 shows that the fairness-purpose allocation of QKnober can guarantee θ-*soft fairness* across users. In the second stage, we perform the efficiency-purpose allocation with the remaining idle resource vector $\mathbf{R}^{'}$ so that its overall efficiency is maximized.

Formally, our work is to search a feasible allocation $\mathbf{U}^{'}$ such that Formula (7) is maximized. Particularly, for any two users i and j with the same normalized task demands (i.e., $\frac{\mathbf{D}_i}{|\mathbf{D}_i|} = \frac{\mathbf{D}_j}{|\mathbf{D}_j|}$), exchanging resources between them has no impact on efficiency but could affect fairness. In order for better fairness, we still keep Formula (9) holding for any two users satisfying $\frac{\mathbf{D}_i}{|\mathbf{D}_i|} = \frac{\mathbf{D}_j}{|\mathbf{D}_j|}$ by adding Formula (19). We can model the efficiency-purpose allocation as a linear programming optimization problem as follows:

$$\textbf{Maximize} \qquad \epsilon(\mathbf{U}^{'}) = \sum_{i=1}^{n}\{N_i(\mathbf{U}_i^{'}) \cdot \sum_{k=1}^{m} d_{i,k}/r_k\}. \tag{15}$$

subject to:

$$s_i^{'}/w_i = s_j^{'}/w_j. \quad (\mathbf{D}_i/|\mathbf{D}_i| = \mathbf{D}_j/|\mathbf{D}_j|, \forall i,j \in [1,n]). \tag{16}$$

and

$$\sum_{i=1}^{n}\{d_{i,k} \cdot N_i(\mathbf{U}_i^{'})\} \leqslant r_k - \sum_{j=1}^{n} \frac{s_j^{max} \cdot \rho \cdot d_{j,k}}{\max_{1 \leqslant k' \leqslant m} d_{j,k'}/r_{k'}}. \tag{17}$$

for $\forall k \in [1,m]$. By resolving the linear program, the optimal (maximum) value of $\epsilon(\mathbf{U}^{'})$ can be obtained. Finally, the total system efficiency $\epsilon(\mathbf{U})$ can be computed by combining the allocation efficiencies in the two allocation phases.

To summarize, we have shown that QKnober is a knob-based fairness-efficiency scheduling policy that can maximize the system efficiency while guaranteeing the θ-*soft fairness* with the provided knob ρ. Particularly, different configurations of the fairness knob ρ can result in different soft fairness guarantees for QKnober (i..e, QKnober is *sensitive* to the fairness degradation under different knobs).

2) Properties Analysis of QKnober. We give an analysis of the three essential properties defined in Section 2 for QKnober.

Theorem 2. (Sharing Incentive): *The QKnober allocation policy is sharing incentive when*

$$\rho \geqslant (\max_{1 \leqslant k \leqslant m}\{\frac{1}{r_k} \cdot \sum_{j=1}^{n} \frac{w_j \cdot d_{j,k}}{\max_{1 \leqslant k' \leqslant m}\{\frac{d_{j,k'}}{r_{k'}}\}}\})/\sum_{j=1}^{n} w_j.$$

The proof of Theorem 2 can be found in Appendix B of Technique Report [14].

By properly configuring the knob ρ according to Theorem 2, QKnober can guarantee that each user can schedule at least as the number of tasks as that under exclusively using its own partition of the system resources with no sharing. Next, we show that QKnober is envy-freeness, namely, no user envies the allocation results of any other users under its allocation.

Theorem 3. (Envy Freeness): *Every user under the QKnober allocation prefers its own allocation to others.*

The proof of Theorem 3 can be found in Appendix C of Technique Report [14].

We next show that QKnober produces an efficient allocation under which no user can increase its allocation without decreasing that of other users.

Theorem 4. (Pareto Efficiency): *The QKnober allocation policy is pareto efficient.*

The proof of Theorem 4 is given in Appendix D of Technique Report [14].

5 Implementation of QKnober

In our former discussions of QKnober policy, there are several key assumptions that may not be the case in a real-world computing system. For practical application of QKnober, we need to relax these assumptions by considering complicated and challenging factors for real applications and computing system. In the following, we highlight these challenges and then give our solutions to address them in YARN. Detailed implementation of QKnober can be found in Appendix E of Technique Report [14].

***C1*: Online Users with a Finite Number of Tasks.** In the previous discussions, it has assumed that there are an infinite number of tasks for each user at any time. However, in practice, the tasks of users are arriving over time, implying that the number of tasks per user is generally finite at a time.

Iterative QKnober Approach. We can address this problem through a small modification on QKnober as follows. First, we classify all users into two kinds: *active users* (i.e., with pending tasks) and *inactive users* (i.e., with no pending tasks). The system maintains the list of active users, where an inactive user becomes active whenever there arrives a pending task of it. The system performs QKnober allocation iteratively. In each allocation round, the system uses progressive filling approach to allocates resources to active users based on QKnober until one of them has all its pending tasks scheduled. After that, the active user becomes inactive and will not be considered in the following allocation. The system then starts a new allocation round and repeats the above allocation procedure until there is no active user or no sufficient idle resources that can be allocated.

***C2*: Heterogeneous and Indivisible Tasks.** In QKnober allocation model, we have assumed that tasks are divisible and all the tasks of a user are homogeneous in their resource demands. However, in the real world, it may not be the case. First, the tasks demands of a user are most likely to be diverse (e.g., different demands between map and reduce tasks of a user's MapReduce job). Second, fractional tasks are often not supported and accepted by existing systems (e.g., MapReduce, Spark).

In QKnober, whether to perform fairness-purpose allocation or to do efficient-purpose allocation is determined by the maximum dominant resource share s_i^{max} and knob factor ρ (See Section 4.1). When the demands of all the tasks of a user are homogeneous, the maximum dominant resource share is fixed and can be estimated by Formula (11) for each user. However, in the heterogeneous case, it varies dynamically with the running and new arriving tasks. Moreover, estimating the maximum dominant resource share in such case is *NP-hard*.

Fitness-based Approximation Approach. We propose a heuristic approach based on the First-Fit algorithm [4] as follows. The algorithm first estimates the *current* average resource demand of tasks based on its running and pending tasks for each user. Then, it computes the maximum dominant resource share for each user by using its current average resource demand of tasks with Formula (11). However, in practice, there could be a large number of pending tasks at runtime. It indicates that picking all pending tasks might not reflect the *current* average resource demand of a user. To address it, we instead only consider a certain number of tasks that just fill the remaining idle space of the cloud system. We achieve and update it for current average resource demand by using the First-Fit algorithm dynamically. That is, we count the pending tasks in

the queue order until the cloud system can be filled. Then the current average resource demand can be estimated based on the running tasks and those counted pending tasks.

***C3*: Heterogeneous and Distributed Computing System.** The QKnober allocation model assumes the computing system as a single super-server, which however may not always be the case. A real-world computing system (e.g., Google production cluster, Amazon EC2) generally consists of many heterogeneous servers with different resource capacities connected via a high-speed network. In this case, scheduling tasks efficiently to the computing system is analogous to the *NP-hard* multi-dimensional knapsack problem [5] mentioned above.

Affinity-based Task Scheduling Approach. We develop a heuristic approach for efficiency-purpose allocation by defining *affinity* of a task relative to the system. That is, when there are some idle resources on a server, we first filter out the set of pending tasks that can be accommodated by that server. We then compute the affinity score for each of these tasks, as the dot product between the task's resource demand and the vector of that server's idle resources. The one with the highest affinity score among all these pending tasks is chosen for scheduling.

6 Experimental Evaluation

6.1 Experimental Setup

Hadoop Cluster. We have implemented QKnober in the version of YARN-2.4.0. We deploy the YARN framework in an Amazon EC2 cluster consisting of 60 Amazon EC2 t2.medium instances each with 2 virtual cores and 4 GB memory. We configure 1 instance as master, and the remaining 59 instances as slaves, each of which is configured with $<$2 virtual cores, 4 GB$>$.

Workloads. We run four data-parallel workloads: 1) *Facebook Workload:* It is based on the distribution of jobs sizes and inter-arrival time at Facebook provided by Zaharia et. al. [25]. The workload consists of 100 jobs. It is a mix of large number of small-sized jobs ($1 \sim 15$ tasks) and small number of large-sized jobs (e.g., 800 tasks[1]); 2) *Purdue Workload:* Seven benchmarks (e.g., WordCount, TeraSort, Grep, InvertedIndex, HistogramMovices, Sequence-Count, Self-join) are randomly chosen from Purdue MapReduce Benchmarks Suite [3], with 100G wikipedia data [2] as input data; 3) *Spark Workload:* It is a combination of six algorithms (e.g., PageRank, GaussianMixture, BinaryClassification, Kmeans and Alternating Least Squares (ALS)) using provided example benchmarks; 4) *TPC-H Workload:* To emulate continuous analytic query, such as analysis of users' behavior logs, we ran TPC-H benchmark queries on Hive [1]. 120 GB data are generated with provided data tools.

6.2 Testbed Experimental Results

This section first evaluates the fairness and efficiency of QKnober under different knob values. Then, we compare the performance of QKnober with its alternatives DRF and Tetris. Finally, the overhead evaluation of our QKnober system can be found in Appendix F of Technique Report [14].

[1] We reduce the size of the largest jobs in [25] to have the workload fit our cluster size.

Fairness vs. Efficiency We show in Section 4.2 that QKnober is an elastic knob-based tradeoff allocation policy that allows users to balance the fairness and efficiency flexibly. In this section, we evaluate the impact of different knob values on the fairness and efficiency with the mix of four workloads experimentally. Suppose that there are four users A, B, C, D with the weighted shares of $1:2:3:4$, each running Facebook, Purdue, Spark and TPC-H workloads on the YARN cluster, respectively. With QKnober policy, we can then maximize the system efficiency while guaranteeing the soft fairness. We define a term called *soft fairness degree* to quantify the soft fairness based on Formula (12). The smaller soft fairness degree indicates the better fairness, and vice versa.

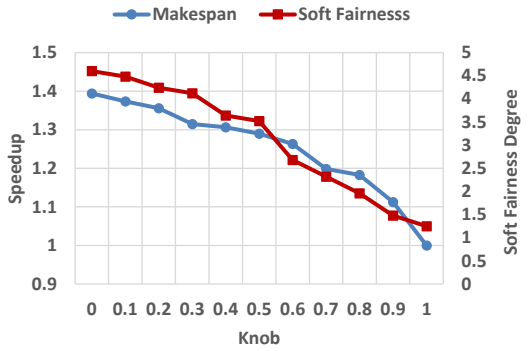
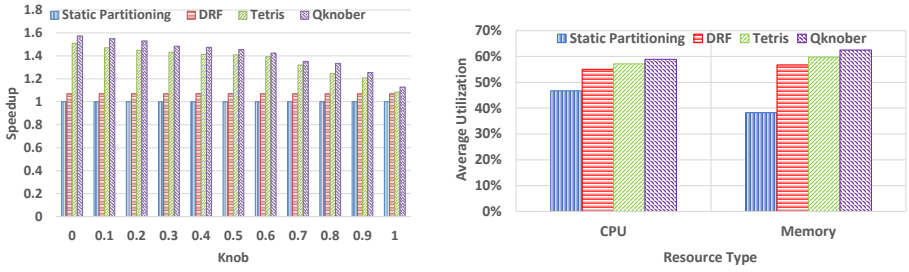


Fig. 3: The system efficiency and soft fairness for QKnober under different knobs, where the speedup is computed over the case of knob $\rho = 1$.

Figure 3 presents the experimental results for QKnober policy under different knob configurations. We compute *speedup* based on the case when the knob is 1.0. The larger value indicates the better performance. It can be observed that there is a strong tradeoff between fairness and efficiency. When the knob is small, it benefits the system efficiency but harms the fairness. In contrast, when we increase the knob value, the fairness can become better at the cost of system efficiency. It means that users can make their own tradeoff preference over the fairness and efficiency by tuning the knob value.



(a) Speedup (over Static Partitioning). (b) Average resource utilization.

Fig. 4: The comparison results of performance and resource utilization for Static Partitioning (i.e., non-sharing case), DRF, Tetris and QKnober under different knob configurations, where the speedup is computed over that of Static Partitioning.

Performance Evaluation Figure 4 (a) gives the performance results for Static Partitioning, DRF, Tetris and QKnober under different knob configurations, where the speedup is computed over that of Static Partitioning. Particularly, we implement the static partitioning policy by dividing the whole cluster resources (e.g., CPU and memory) into four isolated portions for four workloads according to their weights mentioned in Section 6.2, and let them run exclusively without sharing. We have the following observations:

First, resource sharing (e.g., DRF, Tetris and QKnober) performs better than non-sharing (e.g., Static Partitioning). For fairness-only policy DRF, there is about 10% performance improvement over Static Partitioning. In contrast, for fairness-efficiency policies like Tetris and QKnober, the improvement can be up to 57% as we decrease the knob factor ρ from 1.0 to 0.0. The performance gain is mainly due to the resource preemption of unused resources from overloaded users in the sharing case, making the resource utilization higher than the non-sharing case. As illustrated in Figure 4 (b), the resource utilizations for sharing policies (e.g., DRF, Tetris, QKnober) are higher than that of static partitioning. For example, the average cpu utilizations for DRF, Tetris and QKnober are $55\%, 57\%$ and 59%, respectively, higher than the static partitioning of 46%.

Second, QKnober outperforms other baseline allocation policies DRF and Tetris in all knob configurations. Particularly, the reason why QKnober is better than DRF even when the knob is 1.0 is due to its efficient affinity-based task placement in reducing the fragmentation of machines in multi-resource allocation, whereas DRF policy does not have such a concern and simply views all machines as a single super machine. Moreover, both Tetris and QKnober are knob-based fairness-efficiency allocation policies. The reason why QKnober performs better than Tetris is due to their different approaches in the efficiency-purpose allocation. Tetris takes heuristic bin packing approach, whereas QKnober adopts the optimal linear programming method. It makes QKnober achieve a higher resource utilization than Tetris as shown in Figure 4 (b).

7 Related Work

There is a general tradeoff between fairness and efficiency in multi-resource allocation, which has been studied by a lot of research works. Joe-Wong et al. [11] proposed a unifying *mathematical* framework to capture the tradeoff between fairness and efficiency, which are specified by two parameters for a given multi-resource allocation problem. Their work is just a theoretically analytic study and cannot be practically used to real systems such as Hadoop directly. In contrast, our proposed knob-based policy QKnober is practical. We have implemented it in Hadoop that allows users to balance the fairness-efficiency tradeoff flexibly by tuning the knob in the range of $[0, 1]$. Wang et al. [22,24] and Danna et al. [7] studied the fairness-efficiency tradeoff for packet processing consuming both CPU and link bandwidth by proposing a GPS-like fluid model. Tang et al. [17] considered Coupled CPU-GPU architecture by proposing a fairness-efficiency scheduler called EMRF through extending DRF. Wang et al. [21] proposed a bottleneck-aware allocation policy to balance fairness and efficiency for users in multi-tiered storage consisting of SSD and HDD. In contrast, we consider the job scheduling

in cloud computing. Tetris [10] is the most closely related work to our work. It is a fairness-efficiency scheduler for cloud computing that balances the performance and fairness by leveraging alignment heuristics to efficiently pack tasks with heterogeneous resource demands to servers. However, it cannot provide us a soft fairness guarantee given a knob setting due to its *unawareness* of fairness degradation (Section 3) during its fairness-efficiency scheduling. Moreover, it doesn't satisfy sharing incentive property. In comparison, our proposed knob-based policy QKnober is fairness-sensitive, which maximizes the efficiency while guaranteeing the θ-soft fairness under a knob configuration (See Theorem 1). Additionally, it satisfies sharing incentive, envy freeness and pareto efficiency properties with a proper knob configuration.

8 Conclusion

This work studies the tradeoff between fairness and efficiency for users in a shared computing system. Quantifying the fairness degradation/loss is essential for users to better understand the tradeoff. We show that the knob-based approach is a promising solution to achieving the *flexible* and *elastic* tradeoff balance for users. However, existing knob-based fairness-efficiency schedulers are not aware of fairness degradation during its fairness-efficiency allocation, which either fail to guarantee the QoS of δ-fairness or violate desired properties in Section 2. To address it, we develop a new knob-based fairness-efficiency policy called QKnober. It is a fairness sensitive scheduler that allows users to balance the fairness and efficiency with a knob while guaranteeing δ-soft fairness. Typically, we provably show that it meets several desirable properties including sharing incentive, envy freeness and pareto efficiency with a proper knob setting. Finally, we implement QKnober in YARN and our real experiments show that it achieves promised initial results.

Acknowledgement

This work is sponsored by the National Natural Science Foundation of China (61602336, 61772544,U1731125) and Tianjin Natural Science Foundation (18JCZDJC30800).

References

1. Apache tpc-h benchmark on hive. In *https://issues.apache.org/jira/browse/HIVE-600*.
2. Puma datasets. In *http://web.ics.purdue.edu/~fahmad/datasets.htm.*
3. Faraz Ahmad, Seyong Lee, Mithuna Thottethodi, and T. N. Vijaykumar. Puma: Purdue mapreduce benchmarks suite. In *ECE Technical Reports*, 2012.
4. R. P. Brent. Efficient implementation of the first-fit strategy for dynamic storage allocation. *ACM Trans. Program. Lang. Syst.*, 11(3):388–403, July 1989.
5. Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.
6. Benoît Dageville and Mohamed Zait. Sql memory management in oracle9i. In *VLDB '02*, pages 962–973. VLDB Endowment, 2002.
7. E. Danna, S. Mandal, and A. Singh. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *INFOCOM'12*, pages 846–854, 2012.

8. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
9. Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI'11*, pages 323–336, Berkeley, CA, USA, 2011. USENIX Association.
10. Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. In *SIGCOMM'14*, pages 455–466. ACM, 2014.
11. Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework. *IEEE/ACM Trans. Netw.*, 21(6):1785–1798, December 2013.
12. Z. Niu, S. Tang, and B. He. An adaptive efficiency-fairness meta-scheduler for data-intensive computing. *IEEE Transactions on Services Computing*, pages 1–1, 2017.
13. David C. Parkes, Ariel D. Procaccia, and Nisarg Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. *ACM Trans. Econ. Comput.*, 3(1):3:1–3:22, March 2015.
14. Tang Shanjiang, Yu Ce, Sun Chao, Xiao Jian, and Yinglong Li. Qknober: A knob-based fairness-efficiency scheduler for cloud computing with qos guarantees. technical report. In *http://cs.tju.edu.cn/faculty/tangshanjiang/tr/QKnoberTR.pdf*, 2018.
15. S. Tang, B. S. Lee, and B. He. Fair resource allocation for data-intensive computing in the cloud. *IEEE Transactions on Services Computing*, 11(1):20–33, Jan 2018.
16. S. Tang, Z. Niu, B. He, B. S. Lee, and C. Yu. Long-term multi-resource fairness for pay-as-you use computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 29(5):1147–1160, May 2018.
17. Shanjiang Tang, BingSheng He, Shuhao Zhang, and Zhaojie Niu. Elastic multi-resource fairness: Balancing fairness and efficiency in coupled cpu-gpu architectures. In *SC '16*, pages 75:1–75:12, Piscataway, NJ, USA, 2016. IEEE Press.
18. Shanjiang Tang, Bu-sung Lee, Bingsheng He, and Haikun Liu. Long-term resource fairness: Towards economic fairness on pay-as-you-use computing systems. In *ICS '14*, pages 251–260, New York, NY, USA, 2014. ACM.
19. Hal R Varian. Equity, envy, and efficiency. *Journal of economic theory*, 9(1):63–91, 1974.
20. Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, and Agarwal. Apache hadoop yarn: Yet another resource negotiator. In *SOCC '13*, pages 5:1–5:16, New York, NY, USA, 2013. ACM.
21. Hui Wang and Peter Varman. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In *FAST'14*, pages 229–242, Berkeley, CA, USA, 2014. USENIX Association.
22. Wei Wang, Chen Feng, Baochun Li, and Ben Liang. On the fairness-efficiency tradeoff for packet processing with multiple resources. In *CoNEXT '14*, pages 235–248, New York, NY, USA, 2014. ACM.
23. Wei Wang, Baochun Li, and Ben Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *INFOCOM, 2014 Proceedings IEEE*, pages 583–591, April 2014.
24. Wei Wang, Shiyao Ma, Bo Li, and Baochun Li. Coflex: Navigating the fairness-efficiency tradeoff for coflow scheduling. In *INFOCOM'17*.
25. Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys'10*, pages 265–278, New York, NY, USA, 2010. ACM.
26. Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Hot Cloud'10*, volume 10, page 10, 2010.