This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing



# An Adaptive Efficiency-Fairness Meta-scheduler for Data-Intensive Computing

Zhaojie Niu, Shanjiang Tang, Bingsheng He

**Abstract**—In data-intensive cluster computing platforms such as Hadoop YARN, efficiency and fairness are two important factors for system design and optimizations. Previous studies are either for efficiency or for fairness solely, without considering the tradeoff between efficiency and fairness. Recent studies observe that there is a tradeoff between efficiency and fairness because of resource contention between users/jobs. By leveraging the existing schedulers, a meta-scheduler is able to dynamically choose one of them for job/task scheduling at runtime. In this paper, we propose a meta-scheduler called *FLEX* to realize the tradeoff between system efficiency and fairness in Hadoop YARN. FLEX combines multiple existing schedulers into a single aggregated view without any modification on the original schedulers. Equipped with these candidate schedulers, FLEX utilizes machine learning approach to adaptively choose the most proper scheduler according to the characteristic of current running workload and user-defined SLA (Service Level Agreement). We implement FLEX in Hadoop YARN. We conduct experiments with real deployment in a local cluster and perform simulation studies with production traces. Experimental results show that the FLEX outperforms the state-of-the-art approach in two aspects: 1) Given a predefined threshold on the fairness loss, the FLEX reduces the makespan by up to 22% and 24% in real deployment and the large-scale simulation, respectively; 2) Given the predefined threshold on the makespan reduction, the FLEX reduces the fairness loss by up to 75% and 73% in real deployment and the large-scale simulation, respectively.

**Index Terms**— meta-scheduling; efficiency-fairness tradeoff; data-intensive; Hadoop YARN

## 1 Introduction

In the current era of "big data", data-intensive computing is a common paradigm in clusters and clouds. A lot of large-scale distributed data processing frameworks have thereby emerged and become popular in recent years, including MapReduce [1], Dryad [2], Mesos [3], Hadoop YARN [4] and Spark [5]. Efficiency and fairness are two important concerns for the system design on those shared environments. Efficiency indicates the efficiency of the resource usage and it is usually measured with the makespan of a set of jobs [6]. Fairness is often used to guarantee the fair resource allocation between different users in the shared environment. There have been a lot of fairness measurement approaches proposed in the previous studies [7], [6], [8], [9]. They are mainly defined from two different aspects: from the performance's aspect [7], [6] and from the resource usage's aspect [8], [9]. Many previous studies focus on either efficiency or fairness without considering the tradeoff between efficiency and fairness [10], [11], [12], [7], [13], [14]. Recent studies have showed that there is a tradeoff between efficiency and fairness due to the resource contention and proposed some bi-criteria optimization algorithms [15], [16], [6], [8]. However, all of these algorithms are heuristics, which fail to address the tradeoff between efficiency and fairness. The reason is that, 1) the efficiency and fairness of different schedulers vary a lot due to their distinct optimization purposes; 2) due to the heterogeneous resource demands of submitted jobs, a static scheduler cannot always achieve the best tradeoff between the efficiency and fairness with the variation of the resource demands of running workload during the computation.


- *Zhaojie Niu is with Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY), Interdisciplinary Graduate School, Nanyang Technological University, Singapore.*
- *Shanjiang Tang is with School of Computer Science and Technology, Tianjin University, China.*
- *Bingsheng He is with School of Computing, National University of Singapore, Singapore.*


Figure 1 shows a multi-resource usage profile of tasks from Google in a data center of 12 thousands of machines based on Google trace [17]. Multi-resource means that the resource allocation is performed in multiple resource types (here, CPU and memory are the two resource types). The position of a circle indicates the CPU and memory resources consumed by tasks. The size of a circle is logarithmic to the number of tasks in the position. It shows that there are significantly heterogeneous demands for tasks on CPU and memory resources. Users have diverse demands on different types of resources and the most needed resource is called the dominant resource [7]. We define a metric (named *complementary degree*) to quantify the complementarity of the resource demands of the workload (see the formal definition in Section 3). Ideally, two workloads are complementary to each other, if they demand different dominant resources. For two workloads, the more complementary their resource demands, the greater the potential for efficiency optimization and the less the potential for the fairness loss sacrificed in the efficiency optimization. To illustrate that different complementary degrees of submitted jobs have significant impact on the efficiency and fairness in the sharing environment, we conduct an experiment with Google trace. Figure 2 shows the makespan reduction and the fairness loss of a efficiency-oriented scheduler for a workload with different complementary degrees (The detailed setup can be found in Section 5). With the increase of complementary degree, the makespan reduction becomes higher, and the fairness loss first increases significantly to a highest point and later decreases after it. When the complementary degree lies between 0.5 and 1, the complementary degree increases, the makespan reduction is

1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

getting higher and the fairness loss is also getting larger. When the complementary degree lies between 1 and 2, the complementary degree increases, the makespan reduction is still getting higher while the fairness loss is getting less. It shows that the efficiency and fairness of a scheduler are sensitive to the variation of the workload.

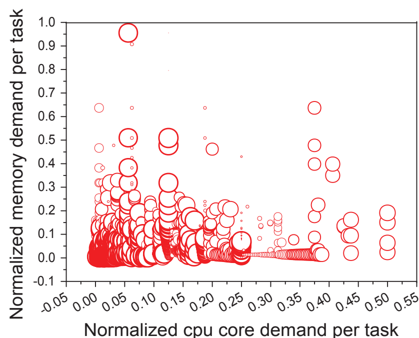



Fig. 1: Heterogeneous resource demand for tasks from Google traces [17]

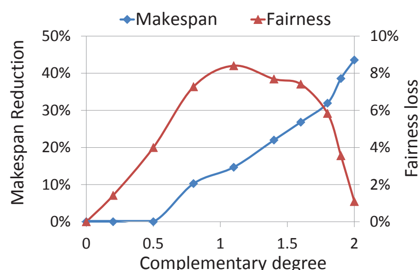



Fig. 2: Tradeoff between efficiency and fairness for workloads with different complementary degrees

Therefore, we should develop a scheduler to be aware of the workload dynamics and efficiency-fairness tradeoff. Since researchers keep inventing new scheduling algorithms, a viable approach is to leverage existing efficiency- and/or fairness-optimized schedulers and develop a meta-scheduler to address the efficiency-fairness tradeoff. Particularly, we propose a meta scheduler called FLEX that takes advantage of existing schedulers in Hadoop YARN and adaptively chooses the most proper scheduler according to the current running workload and the user-defined SLA (Service Level Agreement). FLEX performs the bi-criteria optimization for efficiency and fairness. Given a predefined threshold on fairness loss (i.e., the maximum fairness loss the user can tolerant), FLEX is able to maximize the efficiency of the cluster, or vice versa. FLEX is highly extensible and allows adding/removing any new schedulers on Hadoop YARN. In our current implementation, FLEX supports all mainstream schedulers in the latest Hadoop YARN and one efficiency-oriented scheduler which applies the efficient task packing algorithm proposed in Tetris [6]. Equipping with these candidate schedulers, the FLEX leverages the machine-learning approach to adaptively choose the most proper scheduler with the variation of the current running workload and the user-defined SLA. To support adaptive scheduling, we model the scheduler choosing problem as a well-known classification problem, and resolve the classification problem with decision tree that mainly targets the multi-class classification. Firstly, we train the decision tree model with the data consisting of the scheduling result for different workloads and user-defined SLAs. Then, given the current running workload and user-defined SLA, the target scheduler can be easily inferred based on this built decision tree model.

We implement FLEX in Hadoop YARN (2.6.0). We conduct experiments with real deployment in a local cluster and perform simulation studies with production traces. FLEX performs better than the state-of-the-art scheduling algorithm [6] in two aspects: 1) Given a predefined threshold on the fairness loss, FLEX reduces the makespan by up to 22% and 24% in real deployment and the large-scale simulation, respectively; 2) Given the predefined threshold on the makespan reduction, it reduces the fairness loss by up to 75% and 73% in real deployment and the large-scale simulation, respectively.

The remainder of this paper is organized as follows. Section 2 reviews the background and related work. Section 3 describes the workload characterization model. Section 4 presents our detailed design of FLEX, followed by the experiment results in Section 5. We conclude this paper in Section 6.

## 2 Background and Related Work

### 2.1 Hadoop YARN

The system overview of Hadoop YARN [4] is shown in Figure 3. Hadoop YARN implements major responsibilities of resource management and job scheduling into separate components: *Resource Manager* and per-application *App Master*. The Resource Manager is the unified resource arbitrator among all applications in the system. The *Apps Manager* of Resource Manager launches an App Master for each application which generates resource requests, negotiates resources from the Resource Manager and works with *Node Managers* to execute and monitor the corresponding tasks. Furthermore, Hadoop YARN provides fine-grained resource management instead of coarse-grained slot based manner. Each task is characterized by a *resource requirement vector* which specifies the amount of different resources of multiple types required by this task, e.g., $\langle 1\ CPU,\ 3\ GB \rangle$ indicates 1 CPU core and 3 GB RAM are needed by the task. *YARN Scheduler* of Resource Manager allocates the available resources reported by Node Manager to the pending tasks based on a particular scheduling policy.

There are four mainstream schedulers in Hadoop YARN, including FIFO scheduler, Fair scheduler, Capacity scheduler and DRF scheduler. The FIFO scheduler allocates the resources to applications in first-in-first-out sequence. The Fair scheduler is designed to fairly share the memory among all running users in large-scale multi-tenant clusters. The Capacity scheduler allows YARN applications to run in a multi-tenant cluster and maximizes the throughput of the cluster. It can be considered as a weighted Fair scheduler. DRF scheduler provides the fair allocation of multiple resource types. Besides these schedulers, it is quite easy to integrate new schedulers into Hadoop YARN by implementing the YARN scheduling interface. Current schedulers in Hadoop YARN focus on either the efficiency or the fairness. However, they do not consider the tradeoff between the efficiency and fairness.

### 2.2 Related work

**Efficiency-oriented scheduling.** Maximizing resource utilization is very important for Hadoop. In early years, the early generation of Hadoop abstracts resources into map/reduce slots and allocates them among jobs. DynMR [10] implements more fine-grained reduce tasks with decoupled functional phases in order to resolve



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

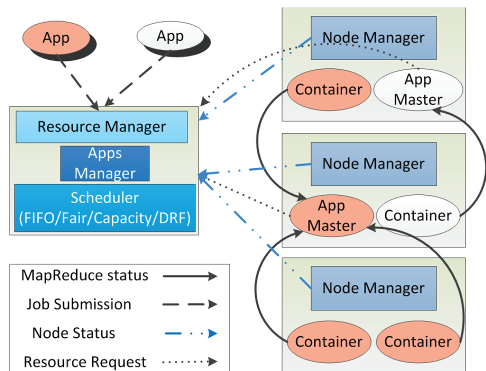


Fig. 3: The overview of Hadoop YARN

the low utilization problem caused by the data skew. RAS [18] captures the heterogeneous resource requirements of workload and dynamically adjusts slots on each machine to maximize the cluster utilization. ILA [11] improves the throughput of the virtual MapReduce clusters by considering the interference between map/reduce tasks. As the development of Hadoop, resource managers in the large-scale cluster are proposed to allocate the resources to the workload in a fine-grained way [3], [4], [12], [19]. They provide a general approach to improve the resource utilization of the cluster by performing coordinated resource allocation and assignment. As the explosive growth of data, I/O optimizations become the core concern of data intensive applications. Delay scheduling achieves nearly optimal data locality by only waiting a small of amount of time [20]. CoHadoop [21] explores more flexible data placement policy to improve the data locality. In addition to these, many more new I/O scheduling algorithms for MapReduce are proposed [22], [23], [24], [25].

**Fair scheduler.** Fair scheduler [26] is proposed in the early generation of Hadoop to allocate slots fairly among different users based on the max-min fairness. Quincy [13] resolves fair allocations efficiently by mapping from the fair scheduling problem to min-cost flow. Choosy [14] is a fair scheduler that considers the fairness with resource constraints in data centers. LTRF [9] resolves the fairness problem in pay-as-you-go environment by considering the historical allocations. Besides the single-resource fair allocation mentioned above, there are a lot of studies for multi-resource fair allocation. Dominant Resource Fairness [7] is the first work to generalize the max-min fairness to multiple resource types on Hadoop YARN. Wang et al. [27] extend Dominant Resource Fairness especially for the heterogeneous environment. Liu et al. [28], [29] propose a novel resource allocation mechanism, called Reciprocal Resource Fairness, to enable fair sharing multiple types of resources in the cloud.

**Efficiency vs. Fairness.** To the best of our knowledge, few studies consider the tradeoff between the efficiency and the fairness on Hadoop YARN. Joe-Wong et al. [16] theoretically analyze the fairness-efficiency tradeoff with multiple resource types for two families of fairness functions. Wang et al. [15], [8] analyze the tradeoff in multi-resource packet processing. Tetris [6] is the first work to explore the tradeoff between efficiency and fairness over YARN framework. Tetris leverages many alignment heuristics to efficiently pack tasks with heterogeneous demands to machines. Although these studies have observed the tradeoff between efficiency and fairness, they are not aware of the variation of the multi-resource demand of the running workload and still perform the scheduling with a static approach during computation. Our preliminary study *Gemini* [30] considers the variation of the workload and proposes a workload-aware scheduling algorithm. This paper extends Gemini in the following major manner. First, we propose a general meta-scheduler which leverages existing schedulers in Hadoop YARN to realize the efficiency-fairness tradeoff. Second, we model the adaptive scheduling as a classification problem and resolve it with decision tree approach.

**Hierarchical Scheduler and Meta-Scheduler design.** Meta Scheduler is a high-level abstract scheduler built atop of a set of specific schedulers like FIFO, Fair schedulers, which has been widely used in Grid, Cloud, HPC and other distributed environments [31], [32], [33], [34], [35], [36], [37], [38], [39]. Computational resources in large-scale Grid are generally managed by a meta-scheduler that interfaces with different specific schedulers to decide the most suitable resources for applications with different preferences [31], [32], [33], [34], [35]. In cloud area, in order to support the coordinated distribution of different cloud workloads, there are some studies proposing some meta schedulers to manage these workloads [36], [37]. Moreover, meta-schedulers are widely used to improve the performance and reduce the energy consumption of HPC systems [38], [39]. Meta-schedulers are also applied in other distributed environments [40]. In contrast, this paper focuses on scheduling data-intensive workloads in a single data center, by improving the resource utilization or the workload fairness, and adaptively choosing the suitable scheduler during the runtime.

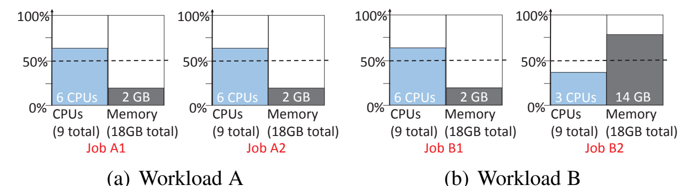
In order to support different applications, researchers have explored design of hierarchical schedulers. YARN [4] and Mesos [3] split the resource management and scheduling between a centralized resource manager and multiple application masters such as Hadoop MapReduce [1] and Spark [5] using an offer-based way. Omega [41], the cluster manager in Google, utilizes optimistic concurrency control to provide high flexibility and parallelism for different workloads that required to access the whole state of the cluster. Rather than focusing on hierarchical scheduler design, this paper is a meta-scheduler focusing on adopting different candidate schedulers on resource management according to workload characteristics and user-defined SLAs.

# 3 Workload Characterization Model and Problem Statement

The meta-scheduler adaptively adopts different candidate schedulers according to the characteristic of the workload and user-defined SLAs. To be aware of the variation of the workload, we propose a resource-based model to characterize the workload and optimize the efficiency and the fairness of the system by adaptively choosing the most proper scheduler at runtime. In this section, we first present our workload characterization model, and then formulate our optimization problem.

## 3.1 Workload characterization model

In this section, we propose an entropy-based approach to calculate the complementary degree of the workload.



(a) Workload A (b) Workload B

Fig. 4: The workloads with different resource demands.



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Entropy is widely used in information theory to characterize the distribution of information content. Larger entropy indicates more random information. We treat the resource demand as the information and then utilize entropy to characterize the distribution of the resource demands of the workload. If the resource demands are randomly distributed, they are complementary for co-scheduling. The randomness of the resource demands indicates the degree of their complementarity. Therefore, we extend the definition of entropy in information theory to quantify the complementary degree of the resource demands of the workload.

We use an example to illustrate the basic concept of complementary degree. We consider two types of resources in our system: CPU and memory. Figure 4 shows two different workloads and each of them consists of two jobs. The job A1 and A2 in workload A are both CPU-intensive. The job B1 and B2 in workload B are CPU-intensive and memory-intensive, respectively. The resource demands of workload A are not complementary to each other as they are all CPU-intensive. In contrast, the resource demands of workload B are complementary to each other because their intensive resources are different. The complementary degree we defined below is used to quantify the complementarity of these resource demands. The complementary degree of the workload B should be higher than the complementary degree of the workload A.

We define some terminology for a multi-resource allocation system. We consider $m$ typed hardware resources (e.g., CPU, memory, disk, network) denoted by $R = \{r_1, ..., r_m\}$. Let $U = \{u_1, ..., u_n\}$ be the set of users sharing the cluster. For every user $i$, let $D_i = (D_{i1}, ..., D_{im})$ be its resource demand vector, where $D_{ij}$ is the fraction of resource $j$ needed by each task of user $i$ over the total capacity of the cluster. For simplicity, we only consider the running tasks and assume the demand for all users are non-negative, i.e., $D_{ij} \geqslant 0, \forall i \in U, j \in R$. We say resource $k_i$ is the dominant resource of user $i$ if

$$k_i = \arg\max_{j \in R} D_{ij}. \tag{1}$$

The dominant resource $k_i$ is the most heavily demanded resource needed by user $i's$ tasks in the resource pool. We calculate the percentage of the users whose dominant resource is $k$ as

$$P(k) = \frac{\sum_{i=1}^{n} \delta(k_i, k)}{n}. \tag{2}$$

$\delta(x, y)$ is an indicator function which is shown as

$$\delta(x, y) = \begin{cases} 1, & \text{if } x = y. \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Statistically, $P(k)$ is the probability of observing a job whose dominant resource type is $k$.

Based on the underlying probability distribution of jobs with different dominant resource types, we quantify the complementary degree of the workload with entropy. According to the definition of entropy [42], the complementary degree $d$ of the workload can be easily calculated as

$$d = -\sum_{i \in R} P(i) \log_2 P(i). \tag{4}$$

### 3.2 Problem statement

The workload characterization model captures the resource usage of different resource types, which essentially reflects the potential of efficiency-fairness tradeoff in the system. Due to the variation of the workload, we need a more fine-grained approach to realize the potential. Based on the workload characterization model, FLEX monitors the variation of the running workload and adaptively chooses the most proper scheduler to perform bi-criteria optimization for efficiency and fairness. In this paper, we evaluate the efficiency with the makespan, i.e., the maximum execution time from the first task submitted till the lasted task completes. For the fairness, following the fairness concept of DRF [7], we consider the fairness of dominant resource. Assume the cluster consists of $n$ hardware resources (e.g., Memory, CPU, Disk) denoted by $R = \{1, ..., n\}$. Let $U = \{1, ..., m\}$ be the set of users sharing the cluster. For each user $i$, let $R_i = (R_{i1}, ..., R_{in})$ be its resource demand vector, where $R_{ir}$ is the share (fraction) of resource $r$ needed by each task of user $i$ during the execution. Resource $r_i^*$ is the dominant resource of user $i$ if

$$r_i^* = \arg\max_{r \in R} R_{ir}. \tag{5}$$

We say the resources are allocated fairly among all users if $r_1^* = r_2^* = ... = r_m^*$. Following DRF and Tetris [6], we evaluate the target scheduler with the improvement on the efficiency and the fairness loss compared with the DRF scheduler. We quantify the improvement on the efficiency with the percentage improvement (or reduction) on the makespan. Let the makespan under the target scheduler is $T^*$ and the makespan under DRF scheduler is $T$, then the makespan reduction $E$ of the target scheduler can be calculated as

$$E = \max\left\{\frac{T - T^*}{T}, 0\right\}. \tag{6}$$

Many fairness definitions are proposed to guarantee the fair resource allocation in the shared environment [7], [6], [9]. As in the previous studies [7], [6], it is very natural and straightforward to measure the fairness along the lifetime of the job with its favored/degraded performance. In our paper, we directly apply this kind of approach. For completeness, we also study the impact the other kind measurement approach which is defined from the resource usage's aspect [9] (more details can be referred in Section 5.2.4).

Many previous studies (e.g., [7], [6]) quantify the fairness from the per-user performance aspect because the execution time of the applications from individual users is the key factor that the users really care about in the shared cluster and the comparison of fairness can be made by considering how the schedulers favor/degrade performance among users (e.g., [6], [43], [44]). Particularly, most of those studies [7], [6] have measured the fairness of the proposed scheduler on the basic of slowdown of each user compared with that of a fair scheduler (such as DRF [7] scheduler and max-min fair scheduler [26]). More precisely, the slowdown refers to the difference in the expected execution time for the same user between when it is scheduled with others under the proposed scheduler and when it is scheduled under a fair scheduler. We directly follow these studies and apply their definitions of fairness in our paper. That is, we quantify the fairness loss of the target scheduler with the average slowdown (reduction) of the job completion time compared with the fair scheduler. Let $J = \{1, ..., k\}$ be the set of users sharing the cluster. For each user $i$, the completion times of its applications under the target scheduler and fair scheduler are $t_i^*$ and $t_i$, respectively. Let



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

$s_i$ be the reduction of the completion time of the applications from user $i$ and $s_i$ can be easily calculated as

$$s_i = \max\left\{\frac{t_i^* - t_i}{t_i}, 0\right\}. \quad (7)$$

Then the fairness loss $F$ of the target scheduler during the whole execution can be calculated as

$$F = \frac{\sum_{i \in J} s_i}{k}. \quad (8)$$

Given the workload characterization model and the definition of efficiency as well as fairness, the optimization problem of this paper is formulated as follows. We consider the bi-criteria optimization between efficiency and fairness in our paper. Particularly, we study two cases for the bi-criteria optimization problem. Given a predefined threshold on the fairness loss $F$, which represents the maximum fairness loss the user can tolerant compared to fair scheduler, FLEX maximizes the makespan reduction $E$ of the system by adaptively choosing the most proper scheduler according to the characteristic of the workload. Similarly, Given the predefined threshold on the makespan reduction $E$, which indicates the makespan reduction expected by the user in comparison with the fair scheduler, FLEX minimizes the fairness loss $F$ of the system.

# 4 FLEX DESIGN

In this section, we introduce the design and implementation of our meta-scheduler FLEX. First, we list a number of rationales used in designing our meta-scheduler FLEX. Second, we give the system overview of FLEX. Third, we describe each component of FLEX in detail. Finally, we show the implementation of FLEX on Hadoop YARN.

## 4.1 Rationale of system design

As we review the related work in Section 2, researchers keep inventing new schedulers for Hadoop/YARN. Ideally, any scheduler reflects some aspect of the efficiency-fairness tradeoff. Thus, a static approach based on heuristics (e.g., [20], [6]) cannot fully address the efficiency-fairness tradeoff. Motivated by the widely applicability of meta-scheduler design in different application environments [35], [36], [38], we propose to design a meta-scheduler to take advantage of the existing or future schedulers in Hadoop YARN. Our system design is driven by a number of rationales.

- Extendability. Since the system targets at a meta-scheduler design, it should be extensible to new schedulers besides the existing schedulers. FLEX should be able to adapt to the removal/addition of a scheduler. Note that, the effectiveness of meta-scheduling still depends on the set of candidate schedulers, and how they can cover the spectrum of the efficiency-fairness tradeoff. The design of our meta-scheduler is able to exploit the spectrum of the efficiency-fairness tradeoff exposed by the candidate schedulers.
- Workload variation awareness. Since the most proper scheduler depends on the characteristic of the workload, the system should provide an automatic mechanism to adaptively switch the scheduler when the workload varies.
- Lightweight runtime overhead. Since the decision has to be made at runtime in order to adapt to workload dynamics, the overhead of the scheduler should be lightweight.

## 4.2 System overview

The overall design of our meta-scheduler FLEX is shown in Figure 5. FLEX integrates multiple existing schedulers in Hadoop YARN into a single aggregated view. These schedulers are candidate schedulers to be chosen by FLEX. FLEX performs the bi-criteria optimization between the efficiency and fairness. Given the user-defined threshold on the fairness loss, the scheduler which satisfies the user-defined SLA and maximizes the efficiency of the cluster is chosen, or vice versa. FLEX implements these optimizations through adaptive scheduling. As the efficiency and fairness of a scheduler depend on the workload, FLEX automatically detects the variation of the current running workload based on the workload characterization model and leverages the machine learning approach to adaptively select the most suitable scheduler from all candidate schedulers. The target scheduler selected from all candidate schedulers becomes the current scheduler of Hadoop YARN and is responsible for the resource allocation till another scheduler is chosen. When the resources in the cluster become available, the current scheduler of Hadoop YARN allocates the resources to the pending jobs/tasks according to its scheduling policy and launches them on the corresponding machine in the cluster.

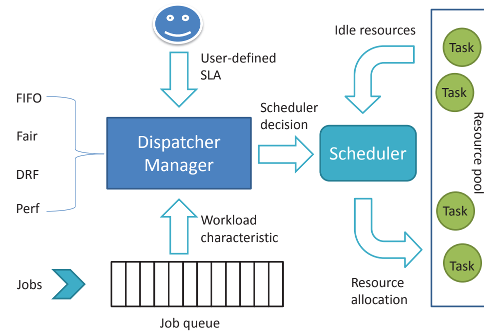


Fig. 5: Design of the meta-scheduler FLEX

## 4.3 Detailed design of FLEX

We introduce the design of FLEX in detail. FLEX consists of a number of candidate schedulers, a dispatcher manager and a scheduler switcher. FLEX integrates existing mainstream schedulers in Hadoop YARN and allows the addition of new schedulers. The dispatcher manager utilizes the machine learning approach to adaptively decide the most suitable scheduler from all candidate schedulers according to the workload characteristics and the user-defined SLAs. Once the target scheduler is decided, the scheduler switcher updates the current scheduler of Hadoop YARN and the chosen scheduler is responsible for the resource allocation till another scheduler is chosen.

***Candidate schedulers***. Essentially, the application scheduling in Hadoop YARN is actually a procedure of determining the job/task execution order of all running applications. All applications are added into a queue and Hadoop YARN determines their execution order according to the policy of current scheduler when resource becomes available. Requests for the first application in the queue are allocated first; once its requests have been satisfied, the next application in the queue is served, and so on. The effectiveness of meta-scheduling in addressing the efficiency-fairness tradeoff depends on the set of candidate schedulers and how they can cover the spectrum of the efficiency-fairness tradeoff. In our current implementation, we consider three mainstream schedulers in Hadoop YARN and add one efficiency-oriented scheduler in our



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

system to demonstrate that new candidate schedulers can be added in a flexible manner.

- **FIFO** scheduler sorts all applications in the order of submission (first in, first out).
- **Fair** scheduler considers the memory usage of all applications and sorts these applications in the decreasing order of how far they are from their fair memory shares.
- **DRF** scheduler takes the amount of dominant resource as the fair share in a system supporting multiple resource types and sorts the applications in the decreasing order of how far they are from their fair shares. Basically, Hadoop YARN has implemented the policy in the original paper [7].
- **Perf** scheduler is an efficiency-oriented scheduler which supports makespan-aware job packing algorithm which is proposed in Tetris [6]. It sorts all applications in the decreasing order of the similarity between their resource requests and the available resources.

***Dispatcher manager.*** With the variation of the running workload and the user-defined SLA, FLEX needs to adaptively choose the most suitable scheduler from all candidate schedulers. We model this decision as a classification problem in the machine learning area. All available candidate schedulers represent a finite set of class labels and the decision of the target scheduler is actually a process of identifying a class label for the target scheduler. Decision tree [45], a well known predictive modeling approach used in machine learning, fits our problem quite well. It creates a tree-structured predictive model which predicts the value of a target item based on the observed features about the item. In the tree-structured model, leaves represent a class or a probability distribution over the classes and the branches represent conjunctions of features that lead to those class labels. Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items. We apply the a recursive greedy algorithm called top-down induction of decision tress (TDIDT) which is by far the most common strategy for learning decision tree from data [46]. We can use different metrics for measuring the splits of the set of items to be classified. These metrics measure the homogeneity of the target label within the subsets. We use two kinds of metrics called *Gini impurity* and *Information gain* in our system. Gini impurity is a measure of the how often a randomly chosen item from the set would be incorrect labeled if it was randomly labeled according to the distribution of labels [47]. Information gain is based information theory and the information gain of an event if the discrepancy of the amount of information before observing that event and the amount after observation [47].

Besides these metrics, there are many other parameters which decide the accuracy and performance of the decision tree model. We describe some important parameters in Table 1 and evaluate the best combinations in the experiment part. Criterion mainly consists of “gini” and “entropy”. The “gini” is the default value due to it is slightly faster compared with entropy. For the max-features, max-depth and max-leaf-node, a smaller value may lead to model inaccuracy, while a too large value will cause overfit on data. By contrast, the decision tree model tends to overfit on data with very small value for the min-samples-split and min-samples-leaf. For the splitter, the “best” strategy is usually applied during the training. In the experiment part, we utilize grid search approach [48] to find the best settings for all the parameters of the decision tree.

| Parameter | Description | Default Value |
|---|---|---|
| Criterion | The function to measure the quality of a split. The “gini” and “entropy” are two supported criteria which represent the gini impurity and the information gain, respectively. | “gini” |
| Splitter | The strategy used to choose the split at each node. The “best” and “random” are two supported strategies which represent the best split and random split, respectively. | “best” |
| max-features | The number of features used in the split. | The number of all features |
| max-depth | The maximum depth of the tree. | 2 |
| min-samples-split | The minimum number of samples which is required to split an internal node. | 10 |
| min-samples-leaf | The minimum number of samples which is required to be at a leaf node. | 5 |
| max-leaf-node | The maximum number of the leaf nodes. If None, then the limitation is ignored. | 10 |
| random-state | the seed used by the random number generator. | np.random |

TABLE 1: Default parameters for the decision tree.

Given the default values for all these parameters which are shown in Table 1, we gain the decision tree, illustrated in Figure 6. The brunch nodes are shown with solid rectangles and the condition expressions of the features inner these rectangles are the rules used for classification. In this example, we mainly consider two features: the complementary degree of the resource demands and the user-defined threshold on the fairness loss. The leaf nodes which are shown with dotted rectangles indicate the classification results. The “Samples” is the total number of items and the vector shows the number of each categories.

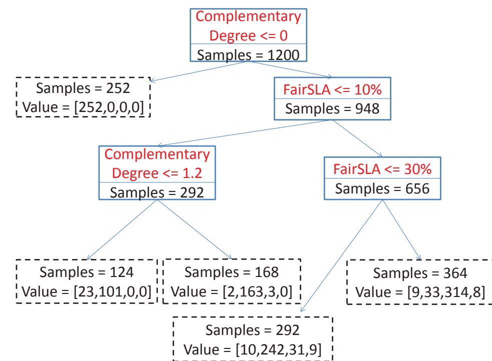



Fig. 6: An example decision tree trained with the default parameter setting

The decision tree can be trained from the labeled training dataset. The training dataset consists of different workloads and user-defined SLAs. Data comes in records of the form $(d,s,Y)$. $d$ is the complementary degree of the workload calculated according to Equation (4). $s$ is the user-defined threshold on the fairness loss or the makespan reduction. $Y$ is the target scheduler we will classify. During the training phase, this variable is already labeled. If our problem is optimizing the efficiency of the cluster given the user-defined threshold on the fairness loss, then $Y$ is the scheduler which satisfies the user's requirement and maximizes the makespan reduction. Similarly, if the our problem is optimizing the fairness between all jobs given the user-defined threshold on the makespan reduction, then $Y$ is the scheduler which can achieve the best fairness under the constraints on efficiency. With the heuristic approach [46], we train the decision tree which can infer



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

the type of $Y$ at runtime according to the workload characteristic $d$ and the user-defined SLA $s$.

Based on the decision tree, the dispatcher manager adaptively decides the most suitable scheduler for Hadoop YARN at runtime. The detail of the decision procedure is shown in Algorithm 1. The scheduler manager calculates the complementary degree when new jobs come or existing jobs finish. Once the complementary degree changes, then we predict the type of the target scheduler among candidate schedulers with the decision tree and update the current YARN scheduler with the decided scheduler. This scheduler is responsible for the resource allocation in Hadoop YARN till another new scheduler is chosen.

**Algorithm 1** Scheduling algorithm

```
1:  c = (FIFO,Fair,DRF,Perf);
2:  s = user-defined SLA;
3:  cur = complementary degree of the current workload;
4:  dt = the decision-tree based predictive model;
5:  YARN = scheduler of Hadoop YARN;
6:  if new jobs come or some jobs finish then
7:      new = complementary degree of the new workload calcu-
    lated according to Equation (4);
8:      if new ≠ cur then
9:          Y = decide the target scheduler from c with dt given
    (new, s);
10:         update the YARN scheduler YARN to Y;
11: allocate resource according to YARN;
```

***Scheduler switcher.*** In order to support the adaptive scheduling of FLEX, we implements a scheduler switcher that is able to dynamically change the scheduling policy of Hadoop YARN at runtime. Once the target scheduler is chosen from the candidate schedulers by the dispatcher manager, FLEX replaces the current scheduler of Hadoop YARN with the chosen scheduler and this new scheduler is responsible for the resource allocation of Hadoop YARN till another scheduler is chosen.

### 4.4 Implementation on Hadoop YARN

We incorporate FLEX into Hadoop YARN (2.6.0) by modifying Resource Manager of Hadoop YARN. The implementation detail is shown in Figure 7. In order to reduce the scheduling latency, Hadoop YARN applies the asynchronous event-based programming model. *AsyncDispatcher* is the core component of the asynchronous programming model. All components of Resource Manager need to register their events dispatchers in the AsyncDispatcher and communicate with each other by sending their events. AsyncDispatcher monitors all coming events and transfers each received event to the corresponding event dispatcher. We incorporate FLEX into YARN framework by making the following modifications:

- *AppsManager* provides a workload query API for other components to gain the information of current running workload including the input data, the application executable, the submission parameters and the resource demand of tasks. When a new job is coming, AppsManager notifies the other components by sending an event *AppsEvent.Start* to AsyncDispatcher. AsyncDispatcher notices WorkloadMonitor that a new application starts by sending an event *MonitorEvent.AppStarted* to it. If this application is completely new, AsyncDispatcher tells ModelTrainer by sending an event *TrainerEvent.NewApp* to it. When a job finishes, AppsManager tells the other components by sending an event AppsEvent.Start to AsyncDispatcher. AsyncDispatcher then notices WorkloadMonitor by sending an event *MonitorEvent.AppFinished* to it.
- Two new components, namely, *ModelTrainer* and *WorkloadMonitor* are integrated into Resource Manager of YARN. Their corresponding event dispatchers are firstly registered in AsyncDispatcher and listen to the corresponding events. ModelTrainer trains the workload characterization model in an offline model using decision tree method. It listens for the *TrainerEvent.NewApp* event and collects the information of the newly coming application from AppsManager. The ModelTrainer retrains and maintains workload characterization model periodically with the latest workloads. In general, the complexity of the decision tree training is $O(mnlogn)$, where $m$ is the number of features and the $n$ is the number of samples in the dataset. In our experiment, the training time of a workload consisting of 1200 jobs is only 1 second, which is ignorable for data-intensive computing. On the other hand, this model training and maintainence can be in an asynchronous/offline manner, which does not interfere the execution of our meta-scheduler. WorkloadMonitor monitors the starting and finishing of all applications and notices DispatcherManager when their resource demands vary.
- We implement two new modules called *DispatcherManager* and *SchedulerSwitcher* in *YARNScheduler* component. All candidate schedulers (FIFO, Fair, DRF and Perf) are integrated into DispatcherManager. DispatcherManager listens for the *SchedulerEvent.DemandVarying* event and provides the adaptive scheduling based on the workload characterization model trained by ModelTrainer. SchedulerSwitcher supports the replacement of the scheduler of Hadoop YARN at runtime.

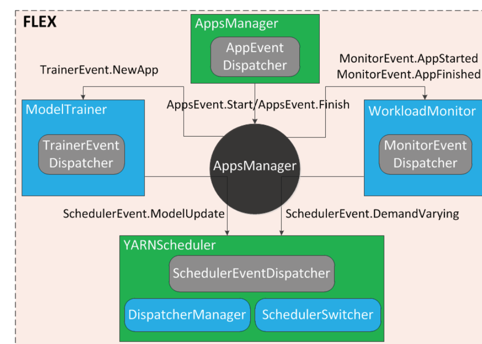



Fig. 7: FLEX implementation on Hadoop YARN. The modified existing components are shown with green rectangles and the newly added components are shown with blue rectangles. Their corresponding event dispatchers are shown with grey round rectangles. The newly added modules of the component are shown with blue round rectangles.

## 5 Evaluation

In this section, we evaluate FLEX with Hadoop YARN on a small cluster and with simulations on the real trace.



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

## 5.1 Experiment setup

We perform two sets of experiments to evaluate FLEX. We evaluate FLEX by running our prototype implementation in our 10-nodes cluster. To evaluate the efficiency-fairness tradeoff and study the parameter impacts at large scale, we conduct a trace-driven simulations using the production trace in Google.

**Hadoop cluster.** We use Hadoop YARN (2.6.0) and run the experiments in our local cluster. The local cluster consists of 10 compute nodes, each with two Intel X5675 CPUs (6 CPU cores per CPU with 3.07 GHz), 24GB DDR3 memory and 500GB 7200RPM disk drivers. These machines are connected with 10Gb/sec Ethernet.

**Workload.** In our experiment, we use two synthesized workloads based on real traces from Facebook and Google.

- Facebook workload. We synthesize a Facebook-like workload based on the distribution of jobs sizes and inter-arrival time at Facebook provided by Zaharia et. al. [20]. The workload consists of 100 jobs. Based on their resource demand, we categorize them into 9 bins according to job types and sizes, as listed in Table 2. It is consisted of large number of small-sized jobs ($1 \sim 15$ tasks) and small number of large-sized jobs (e.g., 800 tasks[1]). The job submission time is derived from one of SWIM's Facebook workload traces (e.g., FB-2009_samples_24_times_1hr_1.tsv) [49]. The demand distribution of map/reduce tasks is based on Figure 1 provided by Ghodsi et al [7]. As YARN currently only supports the allocation of CPU and memory, we also only consider these two resources in real cluster experiments and consider more types of resources in our trace-driven simulation. The actual jobs are from Hive benchmark [50], containing four types of applications, i.e., rankings selection, grep search (selection), uservisits aggregation and rankings-uservisits join.
- Google workload. We also synthesize a Google-like workload by randomly picking 100 jobs from Google trace over a one-hour period.

**Metrics.** We calculate the improvement on the efficiency and the fairness loss of the target scheduler compared with the DRF scheduler. To quantify the improvement on the efficiency, we use the percentage improvement (or reduction) on the makespan, illustrated in Equation 6. For the fairness loss, we calculate it with the average reduction of job completion times, illustrated in Equation 8. There are some other fairness measurement approaches which quantify the fairness from resource aspect [16], [8], [9]. For completeness, we also apply those kinds of approaches and achieve similar results which are shown in Section 5.2.4. In our experiments, we compare our proposed meta-scheduler FLEX with Tetris [6], the state-of-the-art scheduler which studies the tradeoff between the efficiency and the fairness in Hadoop YARN, by showing the reduction on the makespan and fairness loss of FLEX compared to Tetris.

In order to evaluate the accuracy of our decision tree model, we utilize the confusion matrix which is widely used to evaluate the quality of the output of the classifier in machine learning through visualization [51]. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. The diagonal elements of the matrix represent the number of points for which the label is correctly predicted, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions. Based on the confusion matrix, we use three important matrices called accuracy, recall and F1 score. The accuracy is the proportion of the total number of predictions that are correct. The recall is the proportion of the total number of actual instances that are predicted correctly. F1 score is a weighted average of the precision and recall.

**Trace-driven simulator.** In order to evaluate FLEX at a larger cluster, we implement a trace-driven simulator that replays the production traces collected in Google cluster [17]. This trace provides the information of all tasks submitted by over 900 users on a cluster of about 12.5k machines in one month, including task submission times, execution time and normalized CPU/Memory/Disk resource demands. In order to accelerate the simulation, we simulate 60 users submitting tasks with different resource demands for three resource types (CPU, memory and disk) in 24 hours to a 600-node cluster. We assume that all users share the cluster equally.

| Bin | Job Type | Map Tasks | | Reduce Tasks | | # Jobs |
|---|---|---|---|---|---|---|
| | | # | Demand | # | Demand | |
| 1 | rankings selection | 1 | <1,1 GB> | NA | NA | 38 |
| 2 | grep search | 2 | <1, 1.5 GB> | NA | NA | 18 |
| 3 | uservisits aggregation | 10 | <2, 0.5 GB> | 2 | <4,2 GB> | 14 |
| 4 | rankings selection | 50 | <4, 1 GB> | NA | NA | 10 |
| 5 | uservisits aggregation | 100 | <2, 1.5 GB> | 10 | <2, 2 GB> | 6 |
| 6 | rankings selection | 200 | <3, 2 GB> | NA | NA | 6 |
| 7 | grep search | 400 | <2, 1 GB> | NA | NA | 4 |
| 8 | rankings-uservisits join | 400 | <1, 2 GB> | 30 | <2, 0.5 GB> | 2 |
| 9 | grep search | 800 | <2, 0.5 GB> | 60 | <1, 3 GB> | 2 |

TABLE 2: Job types and sizes for synthetic Facebook workloads.

## 5.2 Real deployment evaluations

We evaluate the efficiency and fairness of FLEX with the synthetic workload in our local cluster. We compare FLEX with Tetris. First, we compare their makespan, fairness loss and resource utilizations. Then, we measure the overhead of our scheduling algorithm. Finally, we evaluate the tradeoff model used by FLEX with the cross-validation approach.

### 5.2.1 Overall comparison

We compare the makespan and the fairness loss for FLEX and Tetris. Figure 8(a) shows the makespan reduction of FLEX compared to Tetris for different thresholds on the fairness loss. The makespan reduction is up to 22% and 13% in average. This gain is achieved by considering the variation of the efficiency-fairness tradeoff during the computation. FLEX adaptively decides the suitable scheduler from the candidate schedulers according to the variation of the workload. Instead, Tetris applies the same scheduling policy (*Perf*) through the whole computation which loses many optimization opportunities for efficiency-fairness tradeoff. Similarly, FLEX significantly reduces the fairness loss compared to Tetris for different thresholds on the makespan reduction because FLEX skips the unworthy optimizations which would trade much unnecessary fairness loss for negligible makespan reduction. The result is shown in Figure 8(b). The reduction on the fairness loss is up to 75% and 59% in average compared to Tetris. By designing the adaptive scheduling algorithm, FLEX optimizes the efficiency as well as the fairness at the same time

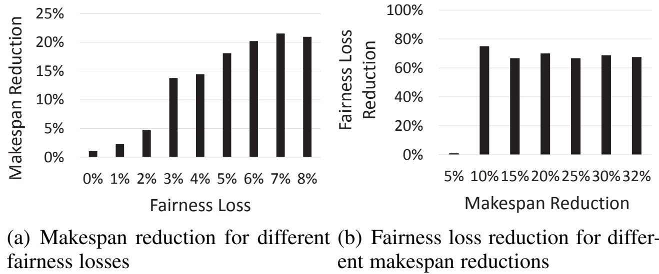
1. We reduce the size of the largest jobs in [20] to have the workload fit our cluster size.



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

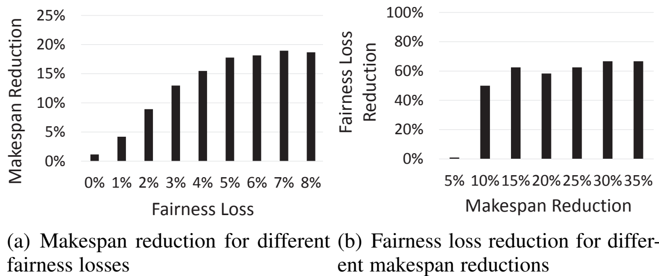
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

compared to Tetris. We also get the similar results with Google workload which are shown in Figure 9(a) and Figure 9(b). The main difference is that the reductions on the makespan and fairness loss for Google workload are both slightly smaller than that for Facebook workload because the resource demands of Google workload are more complementary than the resource demands of Facebook workload.




(a) Makespan reduction for different fairness losses (b) Fairness loss reduction for different makespan reductions

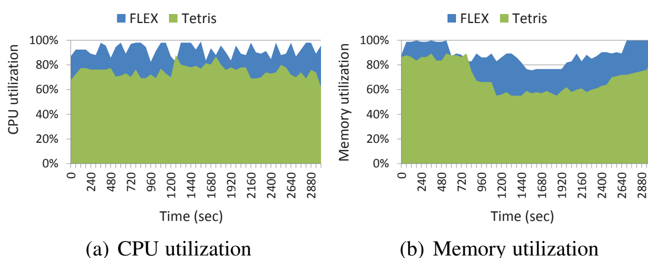
Fig. 8: The reduction on the makespan and fairness loss of FLEX compared to Tetris (Facebook workload)




(a) Makespan reduction for different fairness losses (b) Fairness loss reduction for different makespan reductions

Fig. 9: The reduction on the makespan and fairness loss of FLEX compared to Tetris (Google workload)

To understand the improvement on the efficiency of FLEX compared with Tetris, we compare the resource utilization for both schedulers. As YARN currently only supports the allocation of memory and CPU, we show the utilization of memory and CPU for bother schedulers. In average, FLEX achieves 137% improvement on memory utilization and 122% improvement on CPU utilization. Figure 10(a) shows the detailed CPU utilization of both schedulers during the whole execution when the threshold on the fairness loss is set as 8%. Similarly, the memory utilization of both schedulers is shown in Figure 10(b). The cluster is bottlenecked on different types of resources at different times and FLEX almost fully utilizes the bottlenecked resource all the time. In contrast, Tetris cannot fully utilize both resources due to a large number of resource fragmentation.




(a) CPU utilization (b) Memory utilization

Fig. 10: Resource utilization of the cluster under FLEX and Tetris during the execution

We conduct detailed studies on the impact of all candidate schedulers on the efficiency-fairness tradeoff in Figures 11–14 and have made the following major observations.

First, the schedulers have different optimization goals and the selection of the most suitable scheduler depends on the workload characteristics at runtime. For example, the scheduler which can gain the best efficiency changes with the variation of the workload and user-defined threshold on fairness loss. Figure 11 shows the selection distribution of all candidate schedulers for which can gain the best efficiency for different complementary degrees and different user-defined thresholds on fairness loss. DRF and Perf are mostly used than the other schedulers for all cases. It means that these two schedulers play very important roles in FLEX as we mainly study the efficiency-fairness tradeoff. With the increase of the complementary degree of the workload, the usage of Perf becomes even higher due to more optimizations which satisfies the user-defined SLA on the fairness can be conducted. Similarly, with the increase of the user-defined threshold on fairness loss, the usage of Perf also increases as more efficiency improvement can be achieved with the increase of user's tolerant on the fairness loss.

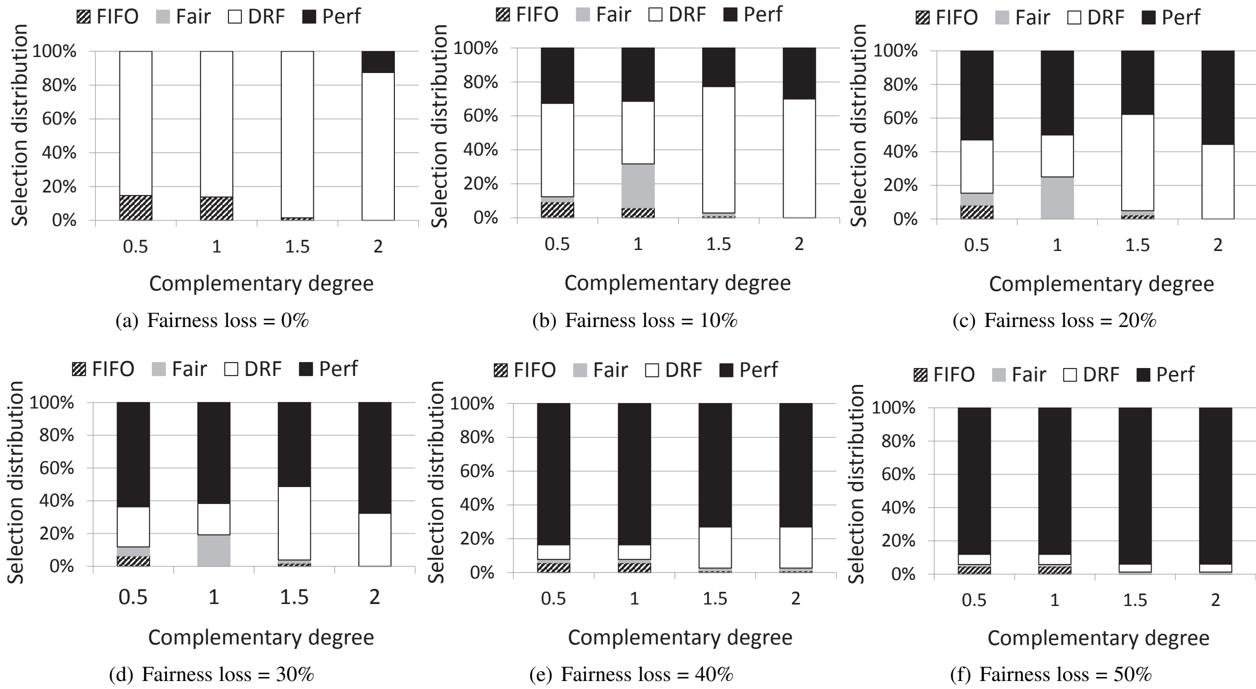
Second, the set of candidate schedulers is very important for the effectiveness of meta-scheduling. Our FLEX considers 4 candidate schedulers in the experiment. In order to show the impact of different combinations of schedulers on the effectiveness of the meta-scheduling, we enumerate all combinations of the candidate schedulers of FLEX and perform the scheduling for different workloads given different user-defined thresholds on fairness losses. We calculate the makespan reduction of each combination compared to DRF scheduler according to Equation 6. If the user-defined threshold on the fairness loss can not be satisfied no matter which scheduler is chosen, we give the penalty on efficiency by setting the makespan reduction to -100%. For each combination, we show the average makespan reduction given different thresholds on the fairness loss for all workloads. We classify these combinations into different categories according to the number of candidate schedulers which can be chosen.

- One scheduler. We assume only one scheduler can be selected. There are 4 combinations in total by choosing only one from these 4 candidate schedulers. Figure 12 shows their average makespan reduction given different thresholds on the fairness loss for all workloads. FIFO, Fair and DRF do not achieve any makespan reduction. For Perf, when the user-defined threshold on fairness loss is large enough, the makespan can be reduced.
- Tow schedulers. There are 6 combinations in total by choosing any two from these 4 candidate schedulers. Figure 13 shows their average makespan reduction given different thresholds on the fairness loss for all workloads. We can see that the combination of DRF and Perf outperforms all the other combinations. DRF guarantees that the user-defined threshold on the fairness can always be satisfied and Perf can reduce the makespan when the user's requirement is satisfied.
- Three schedulers. There are 4 combinations in total by choosing any three from these 4 candidate schedulers. Figure 14 shows their average makespan reduction given different thresholds on the fairness loss for all workloads. Combination (FIFO,DRF,Perf) and combination (Fair,DRF,Perf) achieve similar results and they both outperform than the other combinations. It further validates the conclusion observed before that DRF and Perf play very important role in our meta-scheduling.



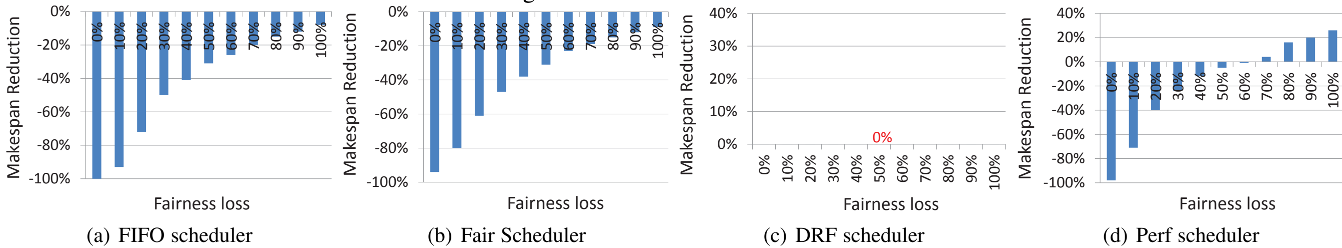
1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing




(a) Fairness loss = 0%
(b) Fairness loss = 10%
(c) Fairness loss = 20%
(d) Fairness loss = 30%
(e) Fairness loss = 40%
(f) Fairness loss = 50%

Fig. 11: The scheduler selection ratio




(a) FIFO scheduler
(b) Fair Scheduler
(c) DRF scheduler
(d) Perf scheduler

Fig. 12: One scheduler (4 combinations in total)

#### 5.2.2 Runtime overhead analysis

In order to evaluate the runtime overhead of our scheduling algorithm, we run experiments with different numbers of jobs and tasks. We evaluate the scheduling overhead by observing the time needed by the Resource Manager (RM) to process the heartbeats coming from Application Masters (AM) and Node Managers (NM). YARN RM conducts the real resource allocation during the NM heartbeat and only updates the resource requests and responses during the AM heartbeat. The processing time of these heartbeats for different schedulers is shown in Table 3. For NM heartbeat, FLEX and Tetris are a bit slower than Hadoop Fair scheduler as they have more complex scheduling logic. For AM heartbeat, they all take the same time. All schedulers perform rather good scalability. We further evaluate the space overhead by monitoring the memory usage on Resource Manager and we find that Gemini consumes almost the same memory as Hadoop Fair scheduler. Our online algorithm design has little runtime overhead, rather than more complex optimizations based on linear programming [52].

| | Hadoop Fair scheduler 10K (50K) tasks | Tetris 10K (50K) tasks | FLEX 10K (50K) tasks |
|---|---|---|---|
| NM heartbeat | .05ms (.18ms) | .08ms (.19ms) | .08ms (.20ms) |
| AM heartbeat | .04ms (.04ms) | .04ms (.04ms) | .04ms (.04ms) |

TABLE 3: Overheads: Average processing time of heartbeats from the Node Manager (NM) and the Application Master (AM) for different schedulers

#### 5.2.3 Model evaluation

We use grid search [48] to find the best settings for all the parameters of the decision tree. We apply these optimal parameter settings which are shown in Table 4 in the following experiments. Based on the optimal setting, we first evaluate the accuracy of the model with the cross validation approach and then study the impact of the prediction error on FLEX.

| Parameter | Best Value |
|---|---|
| Criterion | "entropy" |
| Splitter | "best" |
| max-features | 2 |
| max-depth | 3 |
| min-samples-split | 10 |
| min-samples-leaf | 5 |
| max-leaf-node | 20 |
| random-state | np.random |

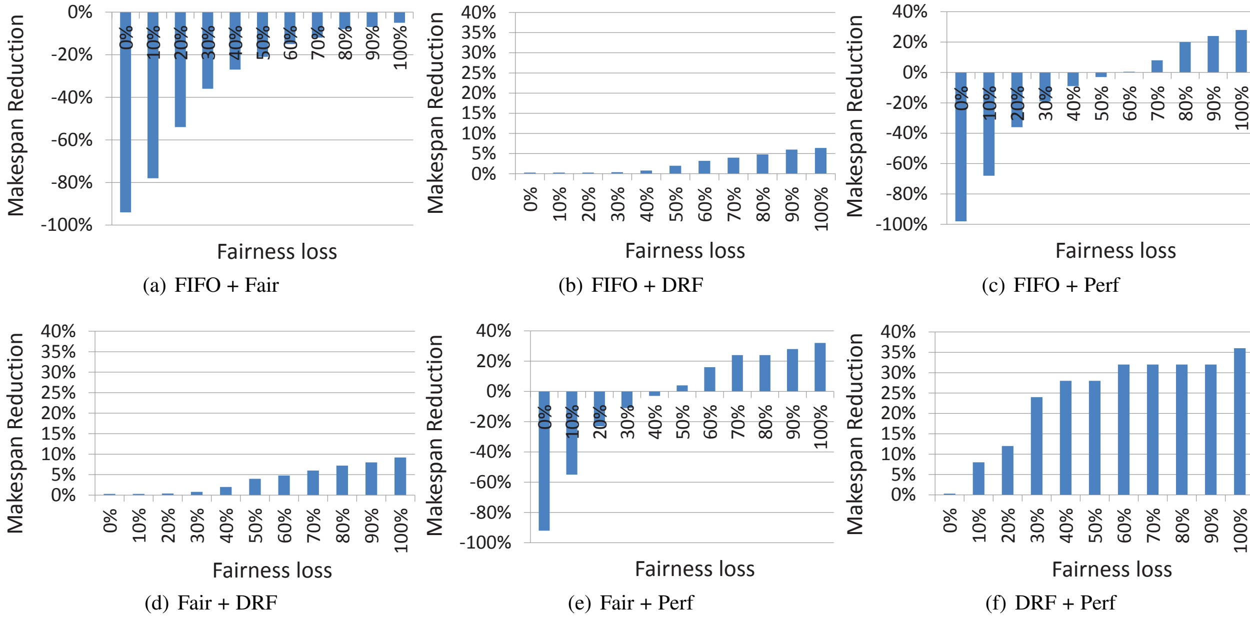
TABLE 4: Optimal parameter setting for the decision tree in the experiment

**Cross validation with the same workload.** We evaluate our decision-tree based tradeoff model with the cross validate approach which is widely used in machine learning. We shuffle the training data and split them into a pair of train and test sets. We use 70% data for training and validate the model with the remaining 30% data. With the parameter setting which is shown in Table 4, we firstly evaluate the accuracy of our decision tree with Facebook workload. The confusion matrix is shown in Figure 15 and the darker color in the diagonal of the confusion matrix indicates



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

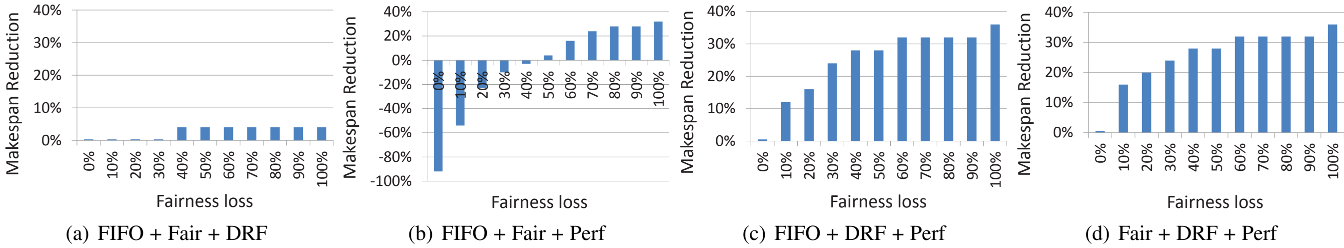
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing




(a) FIFO + Fair (b) FIFO + DRF (c) FIFO + Perf

(d) Fair + DRF (e) Fair + Perf (f) DRF + Perf

Fig. 13: Two schedulers (6 combinations in total)




(a) FIFO + Fair + DRF (b) FIFO + Fair + Perf (c) FIFO + DRF + Perf (d) Fair + DRF + Perf

Fig. 14: Three schedulers (4 combinations in total)

higher accuracy. We also show the detail of related metrics in Figure 16. The results show that our model is accurate enough and can effectively guide the adaptive scheduling of FLEX.

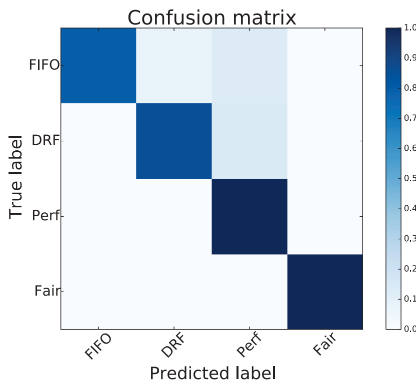



Fig. 15: Confusion matrix (Facebook workload)

**Cross validation with different workloads.** In order to evaluate the robustness of the decision tree model across different workloads, we train our decision tree with Facebook workload and evaluate its accuracy with the Google workload. The confusion matrix is shown in Figure 17 and the related metrics are shown in Figure 18. The average precision, Recall and F1-score are all larger than 80% which further indicates that our model is accuracy even with different workloads.

| | Precision | Recall | F1-score |
|---|---|---|---|
| FIFO | 1.00 | 0.80 | 0.88 |
| DRF | 0.92 | 0.86 | 0.89 |
| Perf | 0.80 | 1.00 | 0.89 |
| Fair | 1.00 | 1.00 | 1.00 |
| avg | 0.93 | 0.91 | 0.91 |

Fig. 16: Classification accuracy of the decision tree (Facebook workload)

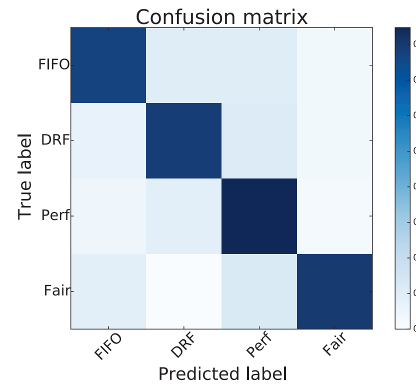



Fig. 17: Confusion matrix (Different workloads)

**Impact of the prediction error on efficiency.** We study the impact of the prediction error of our workload model on the efficiency. Figure 19(a) shows the makespan reduction of



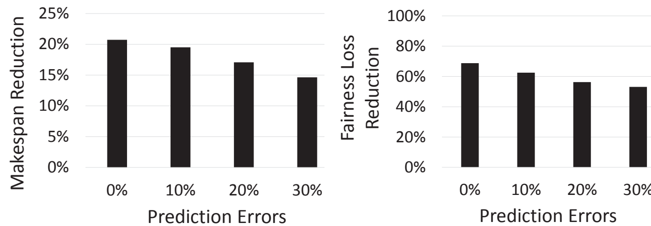
1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

| | Precision | Recall | F1-score |
|---|---|---|---|
| FIFO | 0.80 | 0.77 | 0.78 |
| DRF | 0.81 | 0.79 | 0.80 |
| Perf | 0.72 | 0.85 | 0.78 |
| Fair | 0.90 | 0.80 | 0.85 |
| avg | 0.81 | 0.80 | 0.80 |

Fig. 18: Classification accuracy of the decision tree (Different workloads)

FLEX compared with Tetris by introducing different degrees of prediction errors. The makespan reduction decreases slightly with the increase of the prediction error while our meta-scheduler still reduces the makespan significantly compared with Tetris. Specifically, the allowable threshold on the fairness loss $w$ is 8% here. Given the a prediction error $e$, the prediction has a probability which is randomly distributed in $[w, w(1+e)]$ to randomly choose one scheduler from the candidate schedulers. We vary $e$ from 0% (no error) to 30%. The result demonstrates the robustness of our optimizations, if the prediction error is reasonable.




(a) The makespan reduction for different prediction errors (b) The fairness loss reduction for different prediction errors

Fig. 19: The reduction on the makespan and fairness loss of FLEX compared with Tetris for different prediction errors

**Impact of prediction error on fairness.** We also show the impact of prediction errors on the fairness loss reduction of Flex compared with Tetris in Figure 19(b). The fairness loss reduction slightly decreases with the increase of prediction error while our meta-scheduler still reduces the fairness loss significantly compared with Tetris.

### 5.2.4 Impact of long-term fairness definitions

The fundamental difference between multiple fairness-oriented schedulers is that they have different efficiency-fairness tradeoffs. The choice of different fair schedulers depends on three major factors, including the workload characteristics, the user-defined SLA and the efficiency-fairness tradeoff of the fair schedulers. In fact, our meta-scheduler already integrates two fair schedulers into our system: Fair scheduler and DRF scheduler. Fair scheduler only considers memory. DRF is the state-of-the-art fair scheduler by considering multiple resource types and is treated as the baseline for comparison in our experiment. Figure 12(b) only utilizes the Fair scheduler in our meta-scheduler and Figure 12(c) only considers the DRF scheduler in the meta-scheduler. In Figure 12(b), Fair scheduler violates user-defined fairness loss requirement for some workloads. Therefore, the makespan reduction is penalized and the penalization decreases with the loose of the fairness constraint. In Figure 12(c), the makespan reductions for different thresholds on the fairness loss are all zero as DRF scheduler is treated as the comparison baseline. In Figure13(d), both Fair scheduler and DRF scheduler are integrated into our meta-scheduler and their cooperation actually achieves higher makespan reduction under the fairness constraint compared with the individual scheduler in Figure 12(b) and Figure 12(c). For some workloads, Fair scheduler is selected by the meta-scehduler because it is able to reduce makespan while satisfying the user-defined threshold on the fairness loss. For other workloads, DRF scheduler is chosen by the meta-scheduler to guarantee the user-defined threshold on the fairness loss.

Both Fair and DRF only consider instantaneous resource allocation on a snapshot. In order to evaluate the fairness in one period, we apply the concept of long-term fairness [9] that ensures the fair allocation among multiple users along the time of their execution and the fairness is quantified from resource aspect by considering actual received allocation and purported fair allocation of all applications over their runtime. We make the following observations. First, there is still a tradeoff between the system efficiency and this long-term fairness, which is shown in Figure 20. This tradeoff also depends on the workload that further verifies the importance of the adaptive scheduler in bi-criteria optimization for efficiency and fairness. Second, the fairness loss reduction under the long-term fairness definition is slightly higher compared with that under the definition in Section 5.2.1, due to the interference in the shared environment. In the context of this new fairness definition, FLEX still achieves significant makespan reduction compared to Tetris for the different thresholds on the fairness loss (as shown in Figure 21(a)). Similar, for the different thresholds on the makespan reduction, FLEX also significantly reduces the fairness loss, compared with Tetris (as shown in Figure 21(b)).

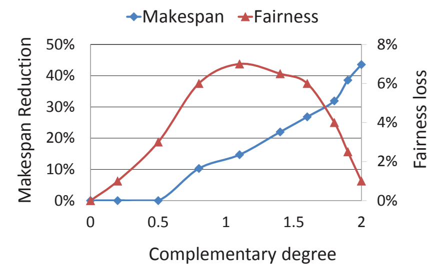
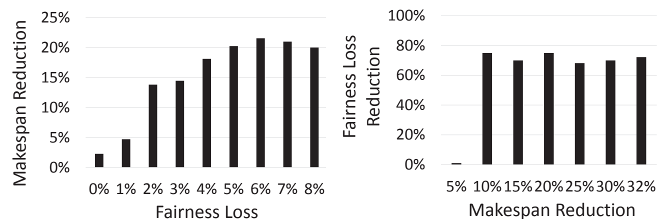



Fig. 20: Tradeoff between the efficiency and fairness of efficiency-oriented scheduler for workloads with different complementary degrees. (long-term fairness)




(a) Makespan reduction for different fair losses (b) Fairness loss reduction for different makespan reductions

Fig. 21: The reduction on the makespan and fairness loss of FLEX compared to Tetris (long-term fairness)

## 5.3 Trace-driven simulations

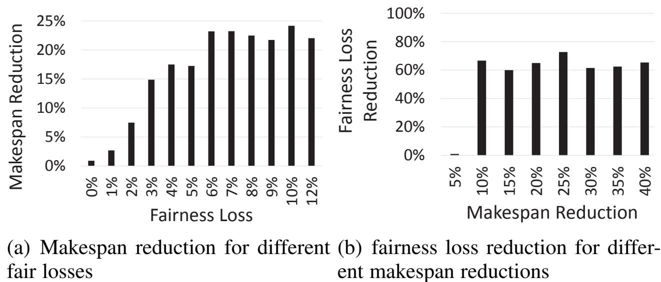
Here, we evaluate the efficiency improvement and fairness loss of FLEX at a larger scale by mimic scheduling in a Google cluster using the production trace provided by Google.

Figure 22(a) shows the makespan reduction of FLEX compared with Tetris for different thresholds on the fairness loss and



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Figure 22(b) gives the result of the reduction on the fairness loss of FLEX compared to Tetris for different thresholds on the makespan reduction. Similar to the results in our local cluster, FLEX can achieve better results than Tetris. We highlight with the following observations for the simulations with the production trace. For the same threshold on the fairness loss in Figure 22(a), the makespan reduction in the simulation is slightly higher than that of the local cluster, because our trace-driven simulator considers more resource types provided in Google trace. Instead, our prototype implementation only considers two resource types as Hadoop YARN currently only supports the allocation of CPU and Memory. This results in more fragmentation and over-allocation of resources.




(a) Makespan reduction for different fair losses

(b) fairness loss reduction for different makespan reductions

Fig. 22: The reduction on the makespan and fairness loss of FLEX compared to Tetris in large-scale simulation with Google trace

## 6 Conclusion

This paper shows that due to the heterogenous demand of multiple resources for users' jobs, being aware of the variation of the resource demand of the running workload is non-trivial for bi-criteria optimization between efficiency and fairness. There is a need to bridge this gap by studying the impact of workload's demand variation on the efficiency and fairness optimizations. In view of this, we propose a meta-scheduler called FLEX to realize the tradeoff between system efficiency and fairness in Hadoop YARN. The experiments on real clusters and simulations show that FLEX achieves better efficiency as well as fairness than the state-of-the-art work.

There are a few interesting studies for extending this work. First, the current system mainly considers the recurring workload. In the future, we are interested in developing more complex online algorithms to support the scheduling of ad-hoc jobs. Second, to further improve resource utilization, we plan to design dynamic and fine-grained resource allocation model by extending the current resource allocation mechanisms provided by Hadoop YARN.

## 7 Acknowledgment

This project is partially funded by a collaborative grant from Microsoft Research Asia and an NUS startup grant in Singapore. Zhaojie's work is in part supported by the National Research Foundation, Prime Ministers Office, Singapore under its IDM Futures Funding Initiative. Shanjiang's work is partly supported by National Natural Science Foundation of China (No.61602336).

## References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *OSDI*, 2004.
[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *EuroSys*, 2007.
[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, 2011.
[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *SoCC*, 2013.
[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012.
[6] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *SIGCOMM*, 2014.
[7] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *NSDI*, 2011.
[8] W. Wang, C. Feng, B. Li, and B. Liang, "On the fairness-efficiency tradeoff for packet processing with multiple resources," in *CoNEXT*, 2014.
[9] S. Tang, B.-S. Lee, B. He, and H. Liu, "Long-term resource fairness: towards economic fairness on pay-as-you-use computing systems," in *ICS*, 2014.
[10] J. Tan, A. Chin, Z. Z. Hu, Y. Hu, S. Meng, X. Meng, and L. Zhang, "Dynmr: Dynamic mapreduce with reducetask interleaving and maptask backfilling," in *EuroSys*, 2014.
[11] X. Bu, J. Rao, and C.-z. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *HPDC*, 2013.
[12] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *Asplos*, 2014.
[13] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *SOSP*, 2009.
[14] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *EuroSys*, 2013.
[15] W. Wang, B. Liang, and B. Li, "On fairness-efficiency tradeoffs for multi-resource packet processing," in *ICDCSW*, 2013.
[16] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *TON*, 2013.
[17] "Google cluster data," https://code.google.com/p/googleclusterdata.
[18] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, "Resource-aware adaptive scheduling for mapreduce clusters," in *Middleware*, 2011.
[19] Z. Niu, B. He, and F. Liu, "Not all joules are equal: Towards energy-efficient and green-aware data processing frameworks," *The IEEE International Conference on Cloud Engineering (IC2E)*, 2016.
[20] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys*, 2010.
[21] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "Cohadoop: flexible data placement and its exploitation in hadoop," in *VLDB*, 2011.
[22] A. Rasmussen, M. Conley, G. Porter, R. Kapoor, A. Vahdat *et al.*, "Themis: an i/o-efficient mapreduce," in *SoCC*, 2012.
[23] S. Ibrahim, H. Jin, L. Lu, B. He, and S. Wu, "Adaptive disk i/o scheduling for mapreduce in virtualized environment," in *ICPP*, 2011.
[24] A. Shieh, S. Kandula, A. G. Greenberg, C. Kim, and B. Saha, "Sharing the data center network." in *NSDI*, 2011.
[25] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *SIGCOMM*, 2011.
[26] "Hadoop mapreduce 1.0 - fair scheduler," http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html.
[27] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *INFOCOM*, 2014.
[28] 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*. IEEE Press, 2014, pp. 970–981.
[29] ——, "F2c: Enabling fair and fine-grained resource sharing in multi-tenant iaas clouds," in *IEEE TPDS*, 2015.
[30] Z. Niu, S. Tang, and B. He, "Gemini: An adaptive performance-fairness scheduler for data-intensive cluster computing," in *IEEE International Conference on Cloud Computing Technology and Science*, 2015.
[31] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

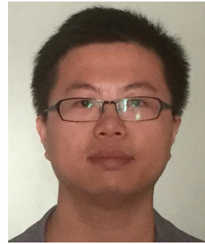


1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2635133, IEEE Transactions on Services Computing

[32] E. Huedo, R. S. Montero, and I. M. Llorente, "A framework for adaptive execution in grids," *Software-Practice and Experience*, vol. 34, no. 7, pp. 631–652, 2004.
[33] S. K. Garg, S. Venugopal, and R. Buyya, "A meta-scheduler with auction based resource allocation for global grids," in *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*. IEEE, 2008, pp. 187–194.
[34] C. Smith, "Open source metascheduling for virtual organizations with the community scheduler framework (csf)," *Technical whitepaper, Platform Computing*, 2003.
[35] K. Chard and K. Bubendorfer, "A distributed economic meta-scheduler for the grid," in *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. IEEE, 2008, pp. 542–547.
[36] A. Foltzer, A. Kulkarni, R. Swords, S. Sasidharan, E. Jiang, and R. Newton, "A meta-scheduler for the par-monad: composable scheduling for the heterogeneous cloud," in *ACM SIGPLAN Notices*, vol. 47, no. 9. ACM, 2012, pp. 235–246.
[37] S. Sotiriadis, N. Bessis, F. Xhafa, and N. Antonopoulos, "From meta-computing to interoperable infrastructures: A review of meta-schedulers for hpc, grid and cloud," in *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*. IEEE, 2012, pp. 874–883.
[38] M. Banikazemi, D. Poff, and B. Abali, "Pam: a novel performance/power aware meta-scheduler for multi-core systems," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*. IEEE, 2008, pp. 1–12.
[39] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 732–749, 2011.
[40] S. Sadhasivam, N. Nagaveni, R. Jayarani, and R. V. Ram, "Design and implementation of an efficient two-level scheduler for cloud computing environment," in *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom'09. International Conference on*. IEEE, 2009, pp. 884–886.
[41] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 351–364.
[42] A. Rrnyi, "On measures of entropy and information," in *Fourth Berkeley symposium on mathematical statistics and probability*, 1961.
[43] H. Arabnejad and J. Barbosa, "Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems," in *ISPA*, 2012.
[44] H. Zhao and R. Sakellariou, "Scheduling multiple dags onto heterogeneous systems," in *Parallel and Distributed Processing Symposium*, 2006.
[45] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.
[46] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.
[47] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, no. 1, pp. 77–93, 2004.
[48] I. Hayashi, T. Maeda, A. Bastian, and L. Jain, "Generation of fuzzy decision trees by fuzzy id3 with adjusting mechanism of and/or operators," in *Fuzzy Systems Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, vol. 1. IEEE, 1998, pp. 681–685.
[49] "Facebook swim trace," https://github.com/SWIMProjectUCB/SWIM.
[50] "Hive performance benchmarks," https://issues.apache.org/jira/browse/HIVE-396.
[51] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.
[52] D. G. Luenberger, *Introduction to linear and nonlinear programming*. Addison-Wesley Reading, MA, 1973, vol. 28.

**Zhaojie Niu** is a Ph.D candidate in the Interdisciplinary Graduate School (IGS), Nanyang Technological University, Singapore. He received his master's and bachelor's degrees from Huazhong University of Science and Technology (HUST), China, in Jan 2012 and July 2009 respectively. He is insterested in big data processing platforms, distributed systems, resource management in the data centers and high-concurrency systems.



**Shanjiang Tang** is an assistant Professor in the School of Computer Science & Technology, Tianjin University. He received his Ph.D degree from Nanyang Technological University in 2015. He received the B.Eng. and M.Sc. degrees from School of Software Engineering and School of Computer Science & Technology at Tianjin University in 2008 and 2011, respectively. His general research interests primarily focus on large-scale computing systems, big data, and cloud computing, with special emphasis on the resource management and job scheduling for Hadoop/YARN system.

**Bingsheng He** received the bachelor degree in computer science from Shanghai Jiao Tong University (1999-2003), and the PhD degree in computer science in Hong Kong University of Science and Technology (2003-2008). He is an Associate Professor in School of Computing, National University of Singapore. His research interests are high performance computing, distributed and parallel systems, and database systems. Since 2010, he has (co-)chaired a number of international conferences and workshops, including IEEE CloudCom 2014/2015 and HardBD2016. He has served in editor board of international journals, including IEEE Transactions on Cloud Computing (IEEE TCC) and IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS).



1939-1374 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.