This document is downloaded from DR-NTU, Nanyang Technological University Library, Singapore.

Title	Gemini: An Adaptive Performance-Fairness Scheduler for Data-Intensive Cluster Computing
Author(s)	Niu, Zhaojie; Tang, Shanjiang; He, Bingsheng
Citation	Niu, Z., Tang, S., & He, B. (2015). Gemini: An Adaptive Performance-Fairness Scheduler for Data-Intensive Cluster Computing. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), 66-73.
Date	2015
URL	http://hdl.handle.net/10220/40532
Rights	© 2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The published version is available at: [http://dx.doi.org/10.1109/CloudCom.2015.52].

Gemini: An Adaptive Performance-Fairness Scheduler for Data-Intensive Cluster Computing

Zhaojie Niu, Shanjiang Tang, Bingsheng He
Nanyang Technological University, Singapore

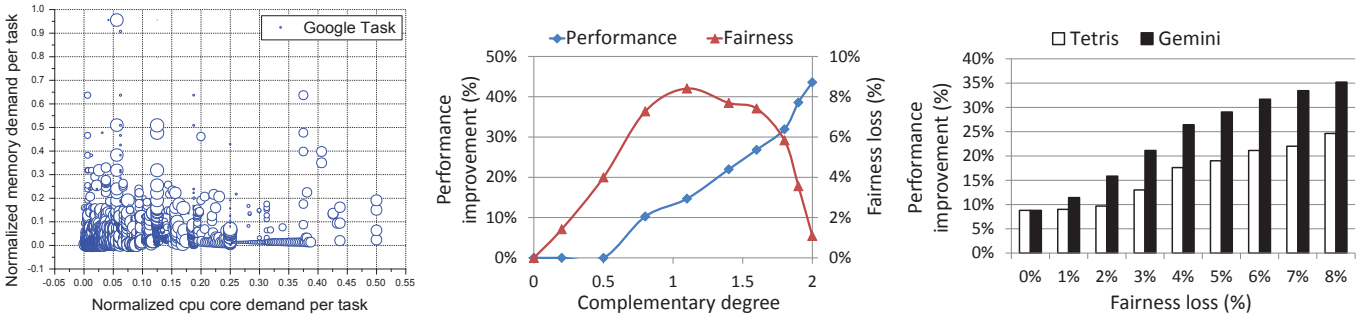
***Abstract*—In data-intensive cluster computing platforms such as Hadoop YARN, performance and fairness are two important concerns for users. Existing studies show that, because of the resource contention between users/jobs, there is a tradeoff between the performance and fairness. In our work, we observe that such trade-off is related to the resource demand of the workload and is changing with the variation of multi-resource demand of submitted jobs during the computation. We also find that having an algorithm to be aware of the resource demand variation is important for the bi-criteria optimization between performance and fairness. However, most previous studies are not aware of this and design their heuristic algorithms with the assumption of fixed trade-off. In this paper, we propose a adaptive scheduler called *Gemini* for Hadoop YARN. For Gemini, it first develops a regression approach to construct a model which can estimate the performance improvement and the fairness loss under the sharing computation compared to the exclusive non-sharing scenario. Next, it leverages the model to guide the resource allocation for pending tasks to optimize the performance of the cluster given the user-defined fairness level. Instead of using a static scheduling policy, Gemini adaptively decides the proper scheduling policy according to the current running workload. We implement Gemini in Hadoop YARN. Experimental results show that Gemini outperforms the state-of-the-art work in two aspects. 1) For the same fairness loss, Gemini increases the performance improvement up to 225% and 200% in real deployment and the large-scale simulation, respectively; 2) For the same performance improvement, Gemini reduces the fairness loss up to 70% and 62.5% in real deployment and the large-scale simulation, respectively.**

I. Introduction

In the current era of "big data", data-intensive cluster computing is a common paradigm in clusters and clouds. A lot of large-scale distributed data processing frameworks have thereby emerged and become more and more popular in recent years, including MapReduce [1], Dryad [2], Mesos [3], Hadoop YARN [4] and Spark [5]. Performance and fairness are two important concerns for users on those shared environments. Previous studies have shown that, due to the resource contention between users/jobs, there is a trade-off between the performance and fairness in optimizations [6], [7], [8], [9], [10]. A number of bi-criteria optimization studies were thereby proposed for the performance and fairness [6], [8]. For example, *Tetris* is a multi-resource fair scheduler for YARN that controls the trade-off between performance and fairness with a knob-based approach [8]. It balances the optimization for performance and fairness with the proposed heuristic job ordering algorithms on the basis of a *fairness knob* takes the values in the range of $[0, 1)$. However, there is an assumption for all of these prior studies that the trade-off between performance and fairness is unchanged. In our work, we observe that, due to the heterogeneous resource demand of submitted workloads, the trade-off between performance and fairness in fact is changing with the variation of the multi-resource demand of submitted jobs during the computation.

Figure 1(a) shows a resource usage profile of tasks from Google in a data center of 12 thousands of machines based on Google trace [11]. 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 varied (heterogeneous) demand for tasks on CPU and memory resources. To illustrate that different *complementary degrees* (defined in Section III) of submitted jobs have significant impact on the performance and fairness in the sharing environment, we conduct an experiment with Google trace. Figure 1(b) shows the performance improvement and the fairness loss for workload with different complementary degrees under the sharing computation compared to the exclusive non-sharing scenario in which the performance is the worst and the fairness is the best. We see that, with the increase of complementary degree, the performance becomes better. For the fairness, in contrast, with the increase of complementary degree, the fairness loss first increases significantly to a highest point and later decreases after it. It means that the tradeoff (i.e., the ratio of performance improvement to the fairness loss) is sensitive to the complementary degree of the workload. To show the importance of being aware of such a varying tradeoff in scheduling, we further show the performance improvement under the same fairness loss for Tetris (i.e., static approach) and our proposed adaptive scheduler called Gemini. We see from Figure 1(c) that our adaptive scheduler achieves a better performance improvement than Tetris under the same fairness loss. The explanation is that, for our scheduler, it can perform the performance-oriented and fairness-oriented resource allocation adaptively according to the complementary degree of running workload during the computation (See Algorithm 1 in Section IV), whereas Tetris not.

Therefore, it motivates us to propose an adaptive scheduler called *Gemini* for Hadoop YARN with the awareness of the impact of submitted workload on the performance improvement and fairness loss. Gemini performs bi-criteria optimization between performance and fairness. Given the user-defined fairness level (i.e., the maximum fairness loss the user can tolerant), Gemini maximizes the performance of the cluster. Gemini firstly leverages the regression approach to



(a) Heterogeneous resource demand for tasks from google traces [11]

(b) Trade-off between performance and fairness for workload with different complementary degrees

(c) Performance improvement under the same fairness loss

Fig. 1: An motivation example with Google trace to show the importance of adaptive scheduling

construct a trade-off model based on the workload information collected from YARN cluster. Given the complementary degree of the workload, this trade-off model can be used to estimate the performance improvement and the fairness loss under the sharing computation compared to the exclusive non-sharing case. Guided by this trade-off model, the resource allocator adaptively decides the proper scheduling policy used in jobs/tasks scheduling in order to maximize the performance of the cluster while the user-defined fairness level is satisfied. Currently, Gemini considers two categories of scheduling policies during runtime, namely, performance-oriented policy and fairness-oriented policy. The performance-oriented policy applies the resource imbalance heuristic which balances the resource capacity left across all resources of the node in order to maximize the resource utilization of the cluster. The fairness-oriented policy applies the dominant resource fairness which allocates resource fairly among users in a system containing different resource types. When resources becomes available, the resource allocator calculates the complementary degree of the current workload and leverage the trade-off model to estimate the performance improvement and the fairness loss. If the estimated fairness loss satisfies the user-defined fairness level and the performance can be improved, the performance-oriented policy is applied. Otherwise, the fairness-oriented policy is used.

We implement Gemini in Hadoop YARN (2.6.0), one of the most popular resource management systems for big data processing. Gemini performs better than the state-of-the-art scheduling algorithm [8] in two aspects. 1) For the same fairness loss, Gemini increases the performance improvement up to 225% and 200% in real deployment and the large-scale simulation, respectively; 2) For the same performance improvement, Gemini reduces the fairness loss up to 70% and 62.5% in real deployment and the large-scale simulation, respectively.

The remainder of this paper is organized as follows. Section II reviews the background and related work. Section III describes the workload characterization model which is used in resource allocation. Section IV presents our detailed design of Gemini, followed by the experiment results in Section V. We conclude this paper in Section VI.

II. Background and Related Work

In this section, we introduce the background of Hadoop YARN and review the related work.

A. Hadoop YARN

Hadoop MapReduce [1] has entered the new generation called Hadoop YARN [4]. The system overview of Hadoop YARN is shown in Figure 2. Compared to a classic Hadoop system, Hadoop YARN features the following major differences in the design. First, Hadoop YARN splits the two major responsibilities of the *JobTracker* in old generation of Hadoop, resource management and job scheduling, into separate components: a *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 the *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 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.

The scheduling policies that are widely used in Hadoop YARN include *Fair*[12], *Capacity*[13] and *FIFO*. The Fair scheduler is designed to fairly share resources among all running applications in large-scale multi-tenant clusters. The Capacity scheduler allows YARN applications to run in a multi-tenant cluster and maximizes the throughput and utilization of the cluster. FIFO scheduler allocates the resources to applications in first-in-first-out sequence. These three schedulers focus on either the performance or the fairness, however, they do not consider the tradeoff between the performance and fairness. In addition to these schedulers, more new schedulers can be integrated into the Hadoop YARN system as plugins.

B. Related work

Performance-oriented scheduling. Many studies are pro-

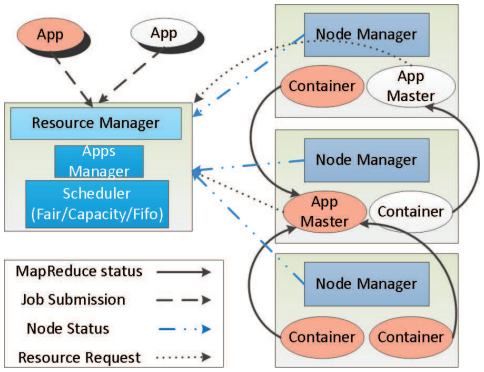


Fig. 2: The overview of Hadoop YARN

posed to optimize the performance of Hadoop. We describe these studies from the following aspects: optimization of resource usages, I/O optimizations and improvement of the programming models. Improving the resource utilizations is very important for Hadoop. In early years, Hadoop 1.0 divides resources into map/reduce slots and allocates them among jobs. DynMR [14] implements more fine-grained Reduce tasks with decoupled functional phases in order to resolve the cluster under utilization problem caused by data skew. RAS [15] captures the heterogeneous resource requirements of workload and dynamically adjusts the slots on each machine to maximize the cluster utilization of the cluster. ILA [16] 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], [17]. 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 [10]. CoHadoop [18] 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 [19], [20], [21], [22]. As the emerges of new applications in Hadoop ecosystems, a lot of new programming models on top of Hadoop are proposed. Incoop [23] optimizes the incremental computations based on MapReduce programming model. HaLoop [24] and Twister [25] propose efficient data processing's for iterative MapReduce programs. Hive [26] provides a SQL-like interface for developers to write queries directly on Hadoop and several studies are developed to optimize the query plans over Hive [27], [28], [29].

Fair scheduler. Fair resource allocation is very important in multi-tenant clusters. In early years, fairness on single resource type attracts a lot of attentions. Fair scheduler [30] is proposed in Hadoop 1.0 to allocate the slots fairly among different users based on the max-min fairness. Quincy [31] resolves fair allocations efficiently by mapping from the fair scheduling problem to min-cost flow. LTRF [32] resolves the fairness problem in pay-as-you-go environment by considering the historical allocations. In order to support multiple resources on Hadoop YARN [4], lots of new studies start to consider the fairness on multiple resource types recently. Dominant Resource Fairness [33] is the first work to generalize the max-min fairness to multiple resource types on Hadoop YARN. Choosy [34] extends Dominant Resource Fairness to support the job placement with constraints in data centers. Wang et al. [35] optimizes the Dominant Resource Fairness especially in the heterogeneous environment.

Performance vs. Fairness. Fruitful studies have been proposed on performance optimization or fairness, however, few studies consider the tradeoff of the performance and the fairness on Hadoop YARN. Tetris [8] is the first work to explore this tradeoff between performance and fairness over YARN framework. Tetris presents a heuristic algorithm to efficiently pack tasks with heterogeneous demand to machines and provides a knob manually configured by the user to tune the trade-off between the performance and the fairness. Wang et al. [9] also analyzed the trade-off in packet processing theoretically. Even though these studies have observed the tradeoff between the performance and the fairness, however, they do not aware the variation of the multi-resource demand of the running workload and perform the scheduling with the assumption of fixed trade-off between performance and fairness during computation. We do extensive study on Hadoop YARN and further propose an adaptive scheduling algorithm which maximizes the performance of the cluster given the user-defined fairness level.

III. Workload Characterization Model

In Figure 1, we show that the resource demand for different tasks are heterogeneous. In order to quantify the complementarity of resource demand for different users, we proposed a notion called *complementary degree*. The more complementary the resource demand, the greater the potential for performance optimization during resource allocation. Figure 1 also shows that both the performance and fairness are sensitive to the complementary degree of the workload. In this section, we propose a model to calculate the complementary degree of the workload and leverage it to characterize the workload.

Entropy is widely used in information theory to characterize the uncertainty of information content. Larger entropy indicates more random information. Inspired by this, we find that it is rather suitable to quantify the complementary degree of the workload with entropy by imaging the resource demand as the information, and differences of these demand correspond to the divergence of the information. The complementarity of resource demand is equivalent to randomness of the information. Therefore, we utilize the definition of entropy to quantify the complementary degree of the workload.

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 required 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

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

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

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

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

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

We quantify the complementary degree of the workload with entropy. According to the definition of entropy [36], the complementary degree d of the workload can be easily calculated as

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

IV. Gemini Scheduler

In this section, we introduce the design and implementation of Gemini. First, we give the system overview of Gemini. Then, we describe the main components of Gemini in detail. Finally, we explain the implementation of Gemini on top of Hadoop YARN.

A. System overview

The logical design of Gemini is shown in Figure 3. It mainly consists of two components, namely, *Model Trainer*, and *Resource Allocator*. Model Trainer collects the workload information from the YARN cluster and leverages the regression approach to construct a trade-off model which can estimate the performance improvement and fairness loss under sharing computation compared to the exclusive non-sharing case. Guided by the trade-off model, Resource allocator adaptively decides the proper scheduling policy according to the complementary degree of current running workload and the user-defined fairness level. Gemini monitors the resource usage of the cluster, when the resources in the cluster becomes available, Resource Allocator allocates the resources to the pending jobs/tasks according to the decided policy and launches them in the cluster on successful allocations.

B. Main components of Gemini

Here, we describe the two main components of Gemini in detail.

Model Trainer. In order to estimate the performance improvement and fairness loss under sharing computation compared to the exclusive non-sharing case, Gemini implements

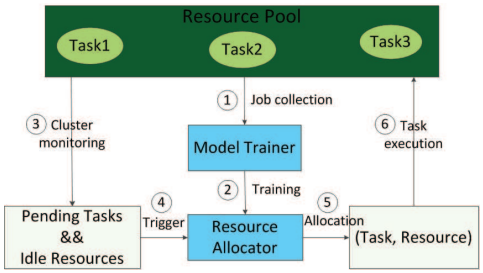


Fig. 3: Logical design of Gemini

an offline component called *Model Trainer*. It leverages the regression approach to construct a trade-off model which can estimate the performance improvement and fairness loss given the complementary degree of the workload. Model Trainer generates the training data by periodically collecting the information of all running workload from the YARN cluster. Model Trainer measures the performance improvement and the fairness loss by scheduling the collected workload under sharing and non-sharing cases, and the complementary degree of the workload can be easily calculated according to Equation (4). Then, Model Trainer uses the measured performance improvement, fairness loss and complementary degree of the workload to construct a trade-off model with regression approach. This trade-off model actually consists of a performance model and a fairness model. Given the complementary degree of the workload, the performance model and the fairness model estimate the performance improvement and fairness loss, respectively. We use a regression approach which is general for both models and can predict the new points efficiently. We use a polynomial which shown as

$$f(d) = c_0 + c_1 * d + ... + c_n * d^n, \tag{5}$$

where $[c_0, c_1, ..., c_n]$ is the coefficients we need to resolve given the training data $(d[i], y[i])$. We train the trade-off model using least-squares and the solution is the coefficients of the polynomial f that minimizes the sum of the squared errors

$$E = \sum_i |y[i] - f(d[i])|^2. \tag{6}$$

With this regression approach, we train the performance model based on the data which is consisted of the complementary degree and the performance improvement. Similarly, we can gain the fairness model. We describe the concrete formulas of these two models in Section V based on the actually measured data in the experiment.

Resource Allocator. Resource Allocator allocates the available resource in the cluster to the jobs/tasks according to a scheduling policy. Instead of using a fixed scheduling policy as the existing work did, Gemini provides mixed policies and adaptively decides the proper scheduling policy based on the resource demand of the running workload during computation. Currently, Gemini supports performance-oriented policy and fairness-oriented policy. The detail of the decision procedure is shown in Algorithm 1. Gemini first calculates the complementary degree of the current running workload in the

cluster according to Equation (4). Then, it leverages the trade-off model which is trained according to Equation (5) (6) to estimate the performance improvement and fairness loss. If the estimated fairness loss satisfies the user-defined fairness level and the performance improvement can be achieved, the performance-oriented policy is applied during resource allocation. Otherwise, Gemini uses the fairness-oriented policy in resource allocation. In our implementation, we consider a heuristic that captures the resource imbalance during resource allocation as the performance-oriented policy, and dominant resource fairness as the fairness-oriented policy. In Gemini implementation, the performance-oriented policy chooses the task whose resource requirement vector can balance the capacity left across all resources which is proven to perform very well in terms of maximizing the resource utilization of the cluster [37]. The fairness-oriented policy represents the current resource usage of a job with the amount of its dominant resource and allocates the resource to the task whose job is furthest from its fair share.

Algorithm 1 Scheduling algorithm

```
1: s = fairness-oriented policy; /*initialize to fairness-oriented policy*/
2: f = user-defined fairness level;
3: w = current running workload;
4: m = trade-off model trained according to Equation (5) (6);
5: calculate complementary degree d of w according to Equation (4);
6: estimate performance improvement i with model m given d;
7: estimate fairness loss l with model m given d;
8: if l ⩽ f and i > 0 then
9:     s = performance-oriented policy;
10: else
11:     s = fairness-oriented policy;
12: allocate resource according to s;
```

C. Implementation on Hadoop YARN

We incorporate Gemini into Hadoop YARN (2.6.0) by modifying Resource Manager of Hadoop YARN. The implementation detail is shown in Figure 4. 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 to AsyncDispatcher. AsyncDispatcher monitors all coming events and transfers each received event to the corresponding event dispatcher. We incorporate Gemini into YARN framework by making the these modifications:

- Apps Manager provides a workload query API for other components to gain the information of current running workload including the input data, the executable, the submission parameters and the resource demand of tasks. When new workload is coming, Apps Manager notifies the other components by sending an event to AsyncDispatcher.
- Two new components, namely, *Model Updater* and *Workload Monitor* are integrated into Resource Manager. Their corresponding event dispatchers are firstly registered in the AsyncDispatcher and listen to the workload update event. Model Updater continuously refines the trade-off model with the newly coming workload and synchronizes the model used by YARN Scheduler. In order not to impact the performance of the YARN cluster and minimize the hardware cost, we scale down the input data size and cluster size and measure the performance improvement and fairness loss in a separate mini-cluster. We set the scale ratio for input data size and cluster size to 10% in Gemini implementation and evaluate the impact of different scale ratios in the Section V. Workload Monitor monitors the running workload and notifies the YARN Scheduler when the resource demand of the workload is varied.
- We integrate the performance-oriented policy and the fairness-oriented policy into Scheduler component, and implement a *Policy Selector* which can adaptively choose the proper scheduling policy based on the resource demand of current workload. The *Resource Allocator* performs the job/task scheduling with the decided policy when resource becomes available.

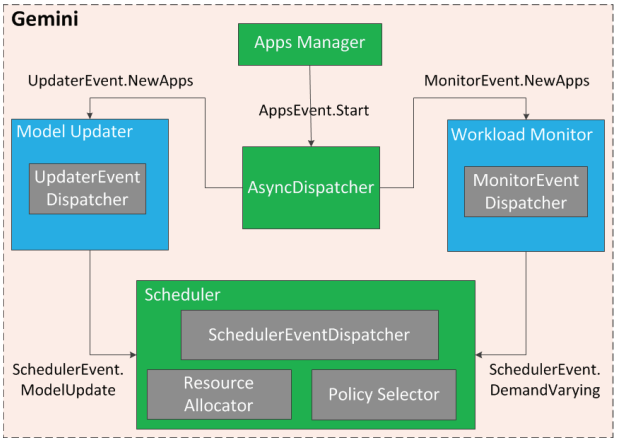


Fig. 4: Gemini implementation on Hadoop YARN. Modifications of existing components in Hadoop YARN are shown with green rectangles. New components added into Hadoop YARN are shown with blue rectangles.

V. Evaluation

A. Experiment setup

We evaluate Gemini by running our prototype implementation in our 10-nodes cluster. To evaluate the performance and study the parameter impacts at large scale, we do a trace-driven simulations using the production trace in Facebook.

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. To test our prototype, we synthesize a Facebook workload based on the distribution of jobs sizes and inter-arrival time at Facebook provided by Zaharia et. al. [10].

The workload consists of 100 jobs. Based on their resource demand, we categorize them into 9 bins of job types and sizes, as listed in Table I. 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]). The job submission time is derived from one of SWIM's Facebook workload traces (e.g., FB-2009_samples_24_times_1hr_1.tsv) [38]. The demand distribution of map/reduce tasks is based on Figure 1 provided by Ghodsi et al [33]. 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 [39], containing four types of applications, i.e., rankings selection, grep search (selection), uservisits aggregation and rankings-uservisits join. We also synthesize a Google workload by randomly picking 100 jobs from Google trace over a one-hour period.

Metrics. We compare the performance and the fairness of a scheduler in the sharing case with the exclusive non-sharing case in which the performance is the worst and the fairness is the best. To quantify the performance improvement, we use the percentage improvement (or reduction) on the makespan. For the fairness loss, we calculate it with the reduction of the average job completion times. In our experiments, we compare our proposed scheduler Gemini with Tetris, the state-of-the-art scheduler which studies the trade-off between performance and fairness in Hadoop YARN.

Trace-driven simulator. In order to evaluate Gemini at a larger cluster, we implement a trace-driven simulator that replays the production traces collected in Google cluster [11]. Google trace provides the information of all jobs in the production cluster including the number of tasks, execution time of these tasks and their normalized resource demand. In order to accelerate the simulation, we use a trace that describes a 7-hour period from the cluster [40].

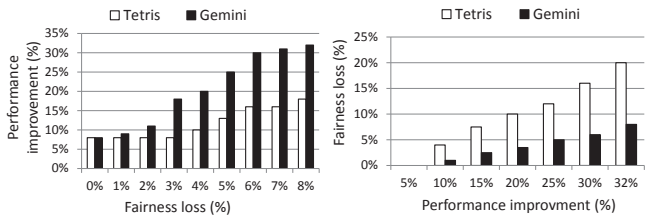
Bin	Job Type	Map Tasks		Reduce Tasks		# Jobs
		#	Demand	#	Demand	
1	rankings selection	1	<1,1 GB>	NA	NA	38
2	grep search	2	<1, 1.5 GB>	NA	NA	18
3	uservisits aggregation	10	<2, 0.5 GB>	2	<4,2 GB>	14
4	rankings selection	50	<4, 1 GB>	NA	NA	10
5	uservisits aggregation	100	<2, 1.5 GB>	10	<2, 2 GB>	6
6	rankings 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 I: Job types and sizes for synthetic Facebook workloads.

B. Real deployment evaluations

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

1) Overall comparison: We compare the performance improvement and the fairness loss for Gemini and Tetris. Figure 5(a) shows the performance improvements of both schedulers under the same fairness loss with the Facebook workload. We see that Gemini achieves better performance compared to Tetris at the cost of the same fairness loss. Noted here, as the maximum fairness loss of Gemini is 8%, Figure 5(a) only shows the performance improvement under the fairness loss which is less than or equal to 8%. The performance improvement is up to 125% and 69.5% in average. This gain is achieved by considering the variation of the trade-off during the computation. Gemini adaptively decides the proper scheduling policy according to the changing of the workload. Instead, Tetris applies the same scheduling policy throughout the whole computation which loses many optimization opportunities even when little or none fairness is lost. Similarly, Gemini achieves better fairness compared to Tetris for the same performance improvement because Gemini skips the unworthy optimizations which would trade huge fairness loss for negligible performance improvement. The result is shown in Figure 5(b). For the same performance improvement, Gemini reduces the fairness loss up to 75% and 55.3% in average compared to Tetris. We see that, by designing the adaptive scheduling algorithm, Gemini optimizes the performance as well as the fairness at the same time compared to Tetris. We also get the similar results with the Google workload. The results are shown in Figure 1(c) and Figure 6.



(a) Performance improvement under the same fairness loss (b) Fairness loss under the same performance improvement

Fig. 5: The comparison results between Tetris and our adaptive scheduler (Facebook workload)

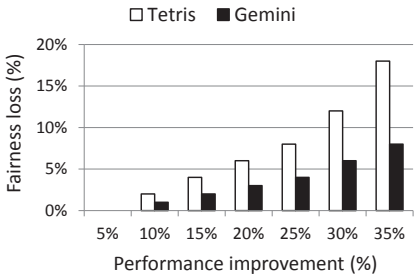
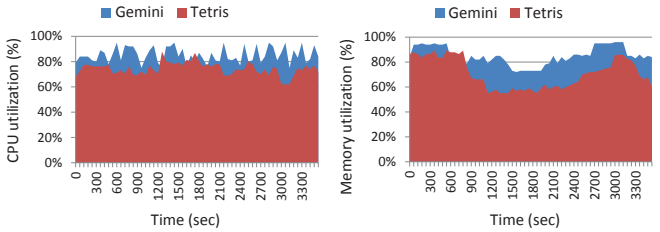


Fig. 6: Fairness loss of different schedulers under the same performance improvement (Google workload)

[1]We reduce the size of the largest jobs in [10] to have the workload fit our cluster size.

To understand the performance improvement of Gemini compared with Tetris on Hadoop YARN, we compare the resource utilization for both schedulers. As YARN currently only supports the allocation of memory and CPU, we only the utilization of memory and CPU. In average, Gemini achieves 137% improvement on memory utilization and 122% improvement on CPU utilization. Figure 7 shows the detailed resource utilization of both schedulers during execution when fairness loss is 8%. We see that the cluster is bottlenecked on different types of resources at different times. In contrast, Tetris is unable to fully use the resources due to fragmentation.



(a) CPU utilization of Gemini and Tetris (b) Memory utilization of Gemini and Tetris

Fig. 7: The cluster resource utilizations of CPU and Memory under Tetris and Gemini during the execution

2) Overhead analysis: In order to evaluate the overhead of our scheduling algorithm, we run experiments with different numbers of jobs and pending tasks. We evaluate the computational 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 does 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 II. For NM heartbeat, Gemini and Tetris are a little slower than Fair 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 Fair. Our online algorithm design has little runtime overhead, rather than more complex optimizations based on linear programming [41].

	Fair 10K (50K) tasks	Tetris 10K (50K) tasks	Gemini 10K (50K) tasks
NM heartbeat	.05ms (.18ms)	.078ms (.19ms)	.08ms (.19ms)
AM heartbeat	.04ms (.04ms)	.04ms (.04ms)	.04ms (.04ms)

TABLE II: Overheads: Average time to process heartbeats from the Node Manager (NM) and the Application Master (AM) for different schedulers

3) Model evaluation: We evaluate our trade-off 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 30% data and the default scale ratio in Gemini is configured to 10%. According to the regression approach which is shown in Equation (5) (6), we gain the concrete formulas for our trade-off model. Given the complementary degree d of the workload, the performance improvement I is calculated as

$$I(d) = -0.00835 + 0.03573d + 0.08681d^2 + 0.00126d^3, \quad (7)$$

and the fairness loss L is calculated as

$$L(d) = -0.00068 + 0.06872d + 0.05256d^2 - 0.04162d^3. \quad (8)$$

The average error for the performance improvement is 8.9% and the average error for the fairness loss is 3.1%. These errors are caused by the incomplete training data and the workload scaling. The previous experiment results show that our model is accurate enough and can effectively guide the resource allocation.

Recall in Section IV-C that Gemini trains the trade-off model by scaling down the workload size as well as the cluster size in order to minimize the hardware cost. We compare the performance improvement and the fairness loss measured after scaling to the values measured in the original environment. Figure 8 shows impact of the scale ratio on the accuracy of our trade-off model. The prediction error decreases with the increase of the scale ratio. In our experiment, we set the scale ratio to 10% by default.

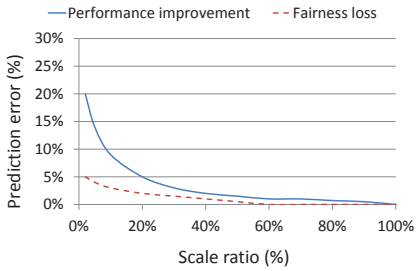


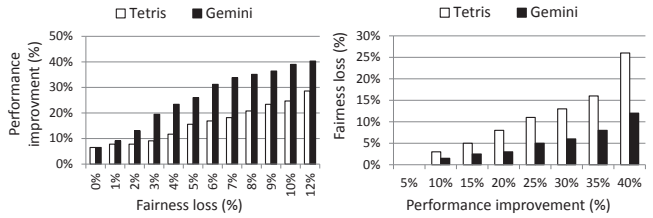
Fig. 8: The relationship between the prediction error and the scale ratio

C. Trace-driven simulations

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

Figure 9(a) shows the performance improvement for both schedulers under different fairness losses and Figure 9(b) gives the result of the fairness loss for both scheduler under different performance improvement. Similar to the results in our local cluster, Gemini can achieves more advantage than Tetris. We highlight with the following observations for the simulations with the production trace. First, for the same fairness loss in Figure 9(a), the performance improvements of both schedulers are slightly larger 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, respectively. Second, for the same performance improvement in Figure 9(b), due to the increase of the total number of jobs, fairness loss of both schedulers both becomes larger than that of the local cluster.



(a) Performance improvements under the same fairness level

(b) Fairness loss under the same performance level

Fig. 9: Overall comparison of different schedulers in large-scale simulation with Google trace

VI. 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 performance and fairness. Thus, it is important to have an adaptive scheduling approach that can perform the performance-oriented and fairness-oriented scheduling at runtime according to the demand complementarity of users' running tasks. However, none of the existing schedulers are aware of the impact of workload's demand variation on the performance and fairness optimizations. In view of this, we present an adaptive scheduler called Gemini which adaptively decides the proper scheduling policy according to the running workload. The experiments on real clusters and simulations show that Gemini achieves better performance as well as fairness than the state-of-the-art work.

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] W. Wang, B. Liang, and B. Li, "On fairness-efficiency tradeoffs for multi-resource packet processing," in *ICDCSW*, 2013.

[7] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *TON*, 2013.

[8] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *SIGCOMM*, 2014.

[9] W. Wang, C. Feng, B. Li, and B. Liang, "On the fairness-efficiency tradeoff for packet processing with multiple resources," in *CoNEXT*, 2014.

[10] 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.

[11] "Google cluster data," https://code.google.com/p/googleclusterdata.

[12] "Hadoop mapreduce next generation - fair scheduler," http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.

[13] "Hadoop mapreduce next generation - capacity scheduler," http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.

[14] 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.

[15] 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.

[16] X. Bu, J. Rao, and C.-z. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *HPDC*, 2013.

[17] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *Asplos*, 2014.

[18] 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.

[19] A. Rasmussen, M. Conley, G. Porter, R. Kapoor, A. Vahdat *et al.*, "Themis: an i/o-efficient mapreduce," in *SoCC*, 2012.

[20] S. Ibrahim, H. Jin, L. Lu, B. He, and S. Wu, "Adaptive disk i/o scheduling for mapreduce in virtualized environment," in *ICPP*, 2011.

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

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

[23] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, "Incoop: Mapreduce for incremental computations," in *SoCC*, 2011.

[24] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: efficient iterative data processing on large clusters," in *VLDB*, 2010.

[25] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *HPDC*, 2010.

[26] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," in *VLDB*, 2009.

[27] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi, "Query optimization for massively parallel data processing," in *SoCC*, 2011.

[28] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in hadoop mapreduce," in *VLDB*, 2012.

[29] R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang, "Ysmart: Yet another sql-to-mapreduce translator," in *ICDCS*, 2011.

[30] "Hadoop mapreduce 1.0 - fair scheduler," http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html.

[31] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *SOSP*, 2009.

[32] 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.

[33] 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.

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

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

[36] A. Rrnyi, "On measures of entropy and information," in *Fourth Berkeley symposium on mathematical statistics and probability*, 1961.

[37] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *SoCC*, 2012.

[38] "Facebook swim trace," https://github.com/SWIMProjectUCB/SWIM.

[39] "Hive performance benchmarks," https://issues.apache.org/jira/browse/HIVE-396.

[40] "Short trace in google cluster," https://code.google.com/p/googleclusterdata/wiki/TraceVersion1.

[41] D. G. Luenberger, *Introduction to linear and nonlinear programming*. Addison-Wesley Reading, MA, 1973, vol. 28.