2015 IEEE 7th International Conference on Cloud Computing Technology and Science

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

Zhaojie Niu
LILY, Interdisciplinary Graduate School
Nanyang Technological University, Singapore

Shanjiang Tang, Bingsheng He
School of Computer Engineering
Nanyang Technological University, Singapore

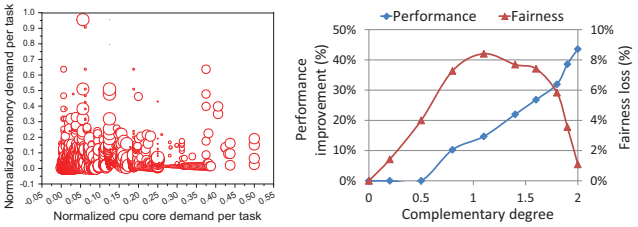
***Abstract*—In data-intensive cluster computing platforms such as Hadoop YARN, performance and fairness are two important factors for system design and optimizations. Many previous studies are either for performance or for fairness solely, without considering the tradeoff between performance and fairness. Recent studies observe that there is a trade-off between performance and fairness because of resource contention between users/jobs. However, their scheduling algorithms for bi-criteria optimization between performance and fairness are static, without considering the impact of different workload characteristics on the tradeoff between performance and fairness. In this paper, we propose an adaptive scheduler called *Gemini* for Hadoop YARN. We first develop a model with the regression approach to estimate the performance improvement and the fairness loss under the sharing computation compared to the exclusive non-sharing scenario. Next, we leverage 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 approach in two aspects. 1) For the same fairness loss, Gemini improves the performance by 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.**

***Keywords*- tradeoff; data-intensive; fairness; performance**

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 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 cluster providers and users on those shared environments. Many previous studies either focus on performance or fairness without considering the tradeoff between performance and fairness [6], [7], [8], [9], [10], [11]. Recent studies have showed that there is a trade-off between performance and fairness due to the resource contention and proposed some bi-criteria optimization algorithms [12], [13], [14], [15]. However, all of these prior studies use a static approach by always applying the same scheduling policy for different workloads. In our work, we observe that, due to the heterogeneous resource demands 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 [16]. 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. We define a metric (*complementary degree*) to quantify such a heterogeneity of the resource demand in a workload (detailed definition can be found in Section III). To illustrate that different complementary degrees 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. With the increase of complementary degree, the performance becomes better, and the fairness loss first increases significantly to a highest point and later decreases after it. It means that the tradeoff between the performance and fairness is sensitive to the complementary degree of the workload.



(a) Heterogeneous resource demand for tasks from Google traces [16]

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

Figure 1: A motivation example with Google trace to show the importance of adaptive scheduling

978-1-4673-9560-1/15 $31.00 © 2015 IEEE
DOI 10.1109/CloudCom.2015.52

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, or vice versa. Gemini firstly leverages the regression approach to 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 to improve the performance of the cluster maximally given that the user-defined fairness level is satisfied. Currently, Gemini considers two scheduling policies during runtime, namely, performance-oriented policy and fairness-oriented policy. The performance-oriented policy applies the resource imbalance heuristic [17] 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 [9] which allocates resource fairly among users in a system containing different resource types. When resources become available, the resource allocator calculates the complementary degree of the current workload and leverages the trade-off model to estimate the performance improvement and the fairness loss. The performance-oriented policy is selected when the estimated fairness loss satisfies the user-defined fairness level and there is an improvement for performance. Otherwise, the fairness-oriented policy is used.

We implement Gemini in Hadoop YARN (2.6.0). Gemini performs better than the state-of-the-art scheduling algorithm [14] 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. 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. Hadoop YARN implements two 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 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 three schedulers in Hadoop YARN, including Fair scheduler, Capacity scheduler and FIFO scheduler. 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.

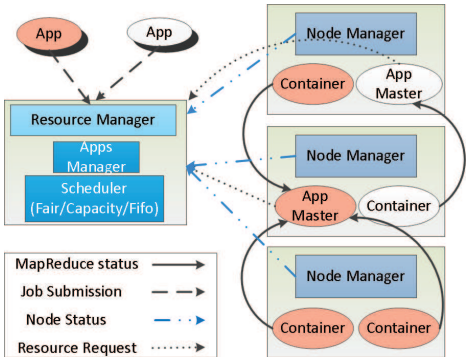


Figure 2: The overview of Hadoop YARN

B. Related work

Performance-oriented scheduling. Many studies are proposed to optimize the performance of Hadoop. We describe these studies mainly from the optimization of the resource usages. 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 [6] implements more

fine-grained reduce tasks with decoupled functional phases in order to resolve 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 [7] 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], [8]. 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 [19]. CoHadoop [20] 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 [21], [22], [23], [24].

Fair scheduler. Fair scheduler [25] is proposed in the early generation of Hadoop to allocate slots fairly among different users based on the max-min fairness. Quincy [10] resolves fair allocations efficiently by mapping from the fair scheduling problem to min-cost flow. Choosy [11] is a fair scheduler that considers the fairness with resource constraints in data centers. LTRF [26] 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 [9] 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.

Performance vs. Fairness. Fruitful studies have been proposed on performance and fairness optimization. However, few studies consider the tradeoff between the performance and the fairness on Hadoop YARN. Joe-Wong et al. [13] theoretically analyzed the fairness-efficiency tradeoff with multiple resource types for two families of fairness functions. Wang et al. [12], [15] analyzed the trade-off in multi-resource packet processing. Tetris [14] is the first work to explore the tradeoff between performance 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 performance and fairness, they do not aware the variation of the multi-resource demand of the running workload and perform the scheduling with a static approach during computation.

III. Workload characterization model

In data-intensive cluster computing, the resource demands for different tasks are heterogeneous as shown in Figure 1. This heterogeneity results in opportunities for bi-criteria optimization between performance and fairness. In order to quantify the complementarity of resource demand for different users, we propose a notion called *complementary degree*. The more complementary the resource demand, the greater the potential for performance optimization during resource allocation. Figure 1(b) 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 for scheduling.

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 demands correspond to the divergence of the information. The complementarity of resource demand is equivalent to randomness of the information. Therefore, we extend 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

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

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

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

$\delta(x,y)$ 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 [30], the complementary degree d of the workload can be easily calculated as

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

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 estimate the performance improvement and fairness loss under sharing computation compared to the exclusive non-sharing case. Guided by the estimation, Resource allocator adaptively decides the proper scheduling policy according to the complementary degree of current running workloads 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.

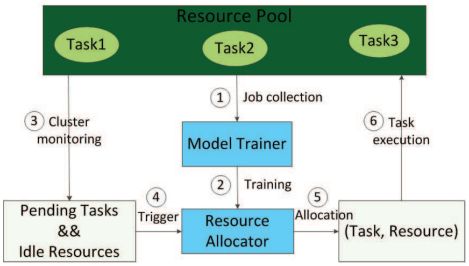


Figure 3: Logical design of Gemini

B. Main components of Gemini

We describe more details for two main components of Gemini.

Model Trainer. In order to estimate the performance improvement and fairness loss in the shared environment compared to the exclusive non-sharing case, Gemini implements 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. Model Trainer generates the training data by periodically collecting the information of all running workloads from the Hadoop 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). Model Trainer uses the measured performance improvement, fairness loss and complementary degree of the workload to construct a trade-off model with the 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. In order to fit the model to curved data, we utilize the polynomial regression.

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

where $[c_0, c_1, ..., c_n]$ is the coefficients we need to resolve and d is the complementary degree of the workload. Given the training data $(d[i], y[i])$ in which $d[i]$ is the complementary degree of workload i and $y[i]$ is the measured performance improvement or fairness loss by running the workload 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

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

We instantiate the concrete formulas of these two models 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 [14], 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 for maximizing the performance given a user-defined fairness level. The algorithm of fairness optimization under a user-defined performance loss is similar and its description is omitted. 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 use Capacity scheduler enhanced by an heuristic that captures the resource imbalance in YARN as the performance-oriented policy and the Fair scheduler in

YARN as the fairness-oriented policy.

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 following modifications:

- Apps Manager 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/task 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 of YARN. 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. 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.

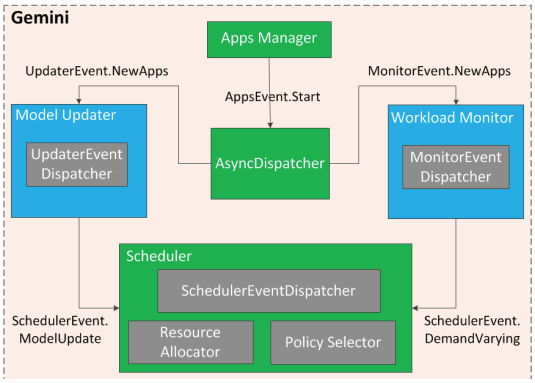


Figure 4: Gemini implementation on Hadoop YARN. Modifications of existing components in Hadoop YARN are shown with green rectangles. New components added into Hadoop YARN are shown with blue rectangles.

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 conduct 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. We synthesize a workload based on the distribution of jobs sizes and inter-arrival time at Facebook provided by Zaharia et. al. [19]. 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 I. 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) [31]. The demand distribution of map/reduce tasks is based on Figure 1 provided by Ghodsi et al [9]. 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 [32], containing four types of applications, i.e., rankings selection, grep search (selection), uservisits aggregation and rankings-uservisits join.

In order to train the performance-fairness model, we measure the performance improvement and the fairness loss used by the model updater in a separate mini-cluster and

[1]We reduce the size of the largest jobs in [19] to have the workload fit our cluster size.

scale down the workload input size accordingly. In our implementation, the scale ratio is set to 10% and we also evaluate the impact of different scale ratios.

Metrics. We compare the performance and the fairness of a scheduler 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 average reduction of job completion times. In our experiments, we compare our proposed scheduler Gemini with Tetris [14], 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 [16]. 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 users share the cluster equally.

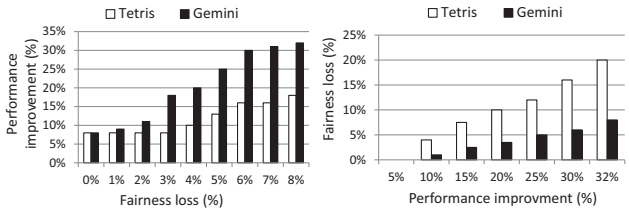
Bin	Job Type	Map Tasks		Reduce Tasks		# Jobs
		#	Demand	#	Demand	
1	rankings selection	1	<1,1 GB>	NA	NA	38
2	grep search	2	<1, 1.5 GB>	NA	NA	18
3	uservisits aggregation	10	<2, 0.5 GB>	2	<4,2 GB>	14
4	rankings selection	50	<4, 1 GB>	NA	NA	10
5	uservisits aggregation	100	<2, 1.5 GB>	10	<2, 2 GB>	6
6	rankings 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 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.

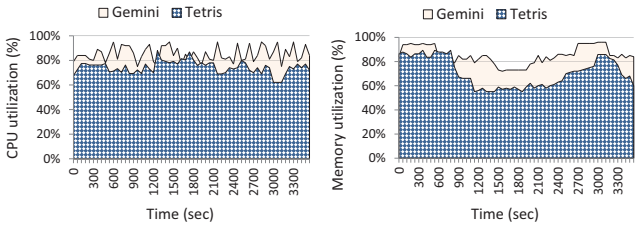
1) Overall comparison: We compare the performance improvement and the fairness loss for Gemini and Tetris. Figure 5(a) shows the performance improvements of both schedulers under the same fairness loss with the Facebook workload. We see that Gemini achieves better performance compared to Tetris at the cost of the same fairness loss. 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.



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

Figure 5: The comparison results between Tetris and our adaptive scheduler

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 6 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 can not fully utilize the resources due to fragmentation.



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

Figure 6: 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 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 II. For NM heartbeat, Gemini 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 [33].

	Hadoop Fair scheduler 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 processing time of 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 derive the concrete formulas for our trade-off model. Given the complementary degree d of the workload, the performance improvement I is calculated as

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

and the fairness loss L is calculated as

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

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 7(a) shows the 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. We also study the impact of the trade-off model on the results. Figure 7(b) shows the performance improvement by introducing different degrees of prediction errors in the fairness loss. Specifically, the allowable fairness loss w is 8%. Given the a prediction error e, the estimation is randomly distributed in $[w, w(1+e)]$. We vary e from 0% (no error) to 30%. The result demonstrates the robustness of our optimizations, if the prediction error is reasonable.

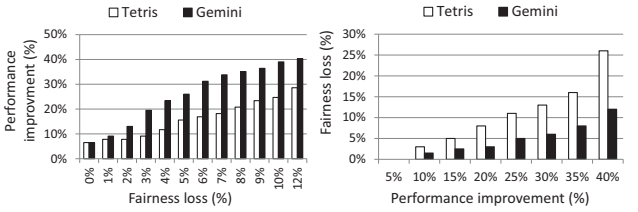
(a) The relationship between the prediction error and the scale ratio (b) The performance improvement for different prediction errors

Figure 7: Sensitivity study of the trade-off model

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 8(a) shows the performance improvement for both schedulers under different fairness loss and Figure 8(b) gives the result of the fairness loss for both schedulers under different performance improvement. Similar to the results in our local cluster, Gemini can achieve better results than Tetris. We highlight with the following observations for the simulations with the production trace. First, for the same fairness loss in Figure 8(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. Second, for the same performance improvement in Figure 8(b), due to the increase of the total number of jobs, the fairness loss of both schedulers is worse than that of the local cluster.



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

Figure 8: 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 varia-

tion 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.

VII. Acknowledgment

This work is supported by a MoE AcRF Tier 1 grant (MOE 2014-T1-001-145) in Singapore. We acknowledge the support from the Singapore National Research Foundation under its Environmental & Water Technologies Strategic Research Programme and administered by the Environment & Water Industry Programme Office (EWI) of the PUB, under project 1002-IRIS-09. Zhaojie's work is in part supported by the National Research Foundation, Prime Ministers Office, Singapore under its IDM Futures Funding Initiative and administered by the Interactive and Digital Media Programme Office.

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] 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.

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

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

[9] 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.

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

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

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

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

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

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

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

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

[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] 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.

[20] 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.

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

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

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

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

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

[26] 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.

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

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

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

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