A Hybrid Scheduling Approach for Scalable Heterogeneous Hadoop Systems

Aysan Rasooli
Department of Computing and Software
McMaster University
Hamilton, Canada
Email: rasooa@mcmaster.ca

Douglas G. Down
Department of Computing and Software
McMaster University
Hamilton, Canada
Email: downd@mcmaster.ca

***Abstract*—The scalability of Cloud infrastructures has significantly increased their applicability. Hadoop, which works based on a MapReduce model, provides for efficient processing of Big Data. This solution is being used widely by most Cloud providers. Hadoop schedulers are critical elements for providing desired performance levels. A scheduler assigns MapReduce tasks to Hadoop resources. There is a considerable challenge to schedule the growing number of tasks and resources in a scalable manner. Moreover, the potential heterogeneous nature of deployed Hadoop systems tends to increase this challenge. This paper analyzes the performance of widely used Hadoop schedulers including FIFO and Fair sharing and compares them with the COSHH (Classification and Optimization based Scheduler for Heterogeneous Hadoop) scheduler, which has been developed by the authors. Based on our insights, a hybrid solution is introduced, which selects appropriate scheduling algorithms for scalable and heterogeneous Hadoop systems with respect to the number of incoming jobs and available resources.**

I. Introduction

Cloud computing promises three distinct characteristics: elastic scalability, pay as you go, and manageability [1]. The advantages of Cloud computing have led to a significant increase in diversity and scale of cloud applications. One of the fastest growing applications is processing Big Data [2]. The scalability and fault tolerance in Cloud computing makes it a great solution for these applications. However, the huge storage and processing requirements of Big Data applications make it extremely challenging to provide the desired performance level for these applications.

Hadoop [3] is a cross-platform framework which supports data intensive distributed cloud applications with a focus on data processing. Hadoop is designed based on the MapReduce [4] programming model. This model divides the applications into several small tasks to be distributed and executed on the Hadoop resources. The goal of Hadoop is to offer efficient and high performance processing of Big Data applications.

A Hadoop scheduler is a critical element to provide the desired performance level for Hadoop users and providers. Schedulers are responsible for assigning the incoming tasks to available resources. However, there are various issues in Hadoop which can directly affect the performance of schedulers, such as heterogeneity and the number of jobs and resources. These issues have been undervalued by most proposed Hadoop schedulers, and as a result can lead to poor performance.

A common enterprise practice is to have a private Hadoop system installed on an intranet. In these Hadoop systems, the jobs and resources may change significantly during a day. As our experiments show, a single scheduling algorithm may not provide the best performance in terms of average completion time. This paper proposes a hybrid approach which uses alternative scheduling algorithms for specific situations. This approach considers average completion time for submitted jobs as the main performance metric. The proposed hybrid scheduler is based on three Hadoop schedulers: FIFO, Fair Sharing [5], and COSHH [6]. The FIFO and Fair Sharing algorithms are the two best known and most widely used Hadoop schedulers. The COSHH algorithm (introduced by the authors of this article), is a Hadoop scheduler which considers the heterogeneity of the system. The hybrid scheduler chooses the best scheduling algorithm for different scales of jobs and resources to address average completion time and fairness.

The remainder of this paper is organized as follows. In Section II, the background for this research is presented. Section III presents important performance issues that Hadoop schedulers should consider for scalable and heterogeneous Hadoop systems. Section IV presents experimental results and analysis of the schedulers. The proposed hybrid solution is introduced and analyzed in Section V. In Section VI, related work is discussed. Finally, conclusions and future work are provided in the last section.

II. Background

Hadoop is a data-intensive cluster computing system, in which incoming jobs are developed using the MapReduce programming model.

A. Hadoop System

A Hadoop cluster is a group of linked resources, where each resource (R_j) has a computation unit and a data storage unit. The computation unit consists of a set of slots, where each slot has a given execution rate. Similarly, the data storage unit has a given capacity and data retrieval rate. Data in the Hadoop system is organized into files, which are usually large. Each file is split into small pieces, which are called slices (usually, all slices in a system have the same size). Hadoop assigns a priority and a minimum share to each user based on a particular policy (e.g. the pricing policy in [7]). The user's minimum share is the minimum number of slots guaranteed for the user at each point in time. The users submit jobs to the system, defined as follows:

- Each job (J_i) in the system consists of a number of *map tasks* and *reduce tasks*.
- A *map task* performs a process on the slice where the required data of the task is located. A *reduce task* processes the results of a subset of a job's *map tasks*. $m(J_i, R_j)$ defines the mean execution time of job J_i on resource R_j.
- Investigations on real Hadoop workloads show that it is possible to classify workloads into classes of "common jobs" [8]. We define the class of jobs (C_k) to be the set of jobs whose mean execution times (on each resource) are the same.

B. Scalability and Heterogeneity in Hadoop Systems

Hadoop heterogeneity can be categorized at three levels: cluster, workload, and users, as follows.

- Workload: incoming jobs are heterogeneous regarding various features such as number of tasks, data and computation requirements, arrival rates, and execution times. Reported analysis on Hadoop systems found their workloads extremely heterogeneous with very different execution times [9]. Moreover, the number of small jobs (with short execution times) exceeds larger size jobs in typical Hadoop workloads such as the Facebook and Cloudera workloads discussed in [9].
- Clusters: resources have different capabilities such as data storage and processing units.
- Users: assigned priorities and minimum share requirements differ between users. Moreover, the type and number of jobs assigned by each user can be different.

Most enterprise Hadoops have a higher load during the day, decreasing during the evening. Similarly, the resource numbers are subject to change at different times. Consequently, using a non-scalable Hadoop system could lead to resource underutilization during evenings or resource overloading and poor performance during peak hours. Scalability in Cloud makes it possible for Hadoop systems to scale up and down based on the load to improve the utilization. A scalable Hadoop considers both job number and complexity as well as the number of available resources to provide sufficient flexibility to adapt. Selecting an appropriate scheduling algorithm for such a scalable heterogeneous Hadoop system is critical to achieve a desired performance level.

C. Hadoop Schedulers

There are various Hadoop schedulers, where each scheduler addresses one or more performance metrics. However, to the best of our knowledge there is no scheduling algorithm which optimizes all these metrics together (see Section II-D below for a list of these metrics). In some cases, optimizing one metric can result in significant degradation in another metric. For instance, a scheduler which optimizes fairness, needs to repeatedly switch the processor between different jobs. This can add significant overhead, which can result in larger average completion times.

To analyze the behaviour of Hadoop schedulers in different Hadoop configurations, this paper uses three Hadoop scheduling algorithms: FIFO, Fair Sharing, and COSHH. The FIFO and Fair Sharing algorithms are used as the basis of a majority of Hadoop schedulers [5, 10, 7, 11]. The COSHH algorithm was first introduced in [6], and considers the system information in making scheduling decisions. Consequently, COSHH is used as a representative of scheduling algorithms which consider heterogeneity at both the resource and workload levels. In the following, we briefly introduce these Hadoop schedulers.

FIFO is the default Hadoop scheduler [4]. It orders the jobs in a queue based on their arrival times, ignoring any heterogeneity in the system. The experience from deploying Hadoop in large systems shows basic scheduling algorithms like FIFO can cause severe performance degradation; particularly in systems that share data among multiple users [1].

Fair Sharing is a Hadoop scheduler introduced to address the shortcomings of FIFO, when dealing with small jobs and user heterogeneity [5]. This scheduler defines a pool for each user, where each pool consists of a number of map and reduce slots on a resource. Each user can use its pool to execute her jobs. If a pool of a user becomes idle, the slots of the pool are divided among other users. This scheduler aims to assign a fair share to each user, which means resources are assigned to jobs such that all users get, on average, an equal share of resources over time.

COSHH[1] is a Hadoop scheduler which considers system and user heterogeneity in making scheduling decisions [6]. Using the system information, COSHH classifies incoming jobs and finds a matching of the job classes to the resources based on the requirements of the job classes and features of the resources. This algorithm solves a Linear Programming problem (LP) to find an appropriate matching of jobs and resources. At the time of a scheduling decision, the COSHH algorithm uses the set of suggested job classes for each resource, and considers the priority, required minimum share, and fair share of users to make a scheduling decision.

D. Performance Metrics

Performance of a Hadoop system can be measured by different metrics, where the five most important ones are as follows:

1) $Average\ Completion\ Time$ is the average completion time of all completed jobs.
2) $Dissatisfaction$ measures how much the scheduling algorithm is successful in satisfying the minimum share requirements of the users.

[1]While the name COSHH was not introduced in [6], it is adopted later.

3) $Fairness$ measures how fair a scheduling algorithm is in dividing the resources among users. A fair algorithm gives the same share of resources to users with equal priority. However, when the priorities are not equal, then a user's share should be proportional to their weight.
4) $Locality$ is defined as the proportion of tasks which are running locally on the resource that contains their stored data. Hadoop systems deal with large data volumes. Consequently, a mapper and reducer which processes Big Data can have significant communication costs if locality is neglected. A *map task* is defined to be local on a resource R, if it is running on resource R, and its required slice is also stored on resource R.
5) $Scheduling\ Time$ is the total time spent for scheduling all of the incoming jobs. This metric measures the overhead of each Hadoop scheduler.

As optimizing one metric can result in significant degradation in the other, it should be noted that optimizing all Hadoop performance metrics together is typically impossible. Therefore, in this paper, we consider the average completion time as the main metric, while considering impact on fairness and overhead of scheduling.

III. Performance Issues

This section analyzes the main drawbacks of each scheduler in a scalable and heterogeneous Hadoop system. For this purpose, this paper uses an example system, which includes four heterogeneous resources and three users with the following characteristics (the choice of system size is only for ease of presentation, the same issues arise in larger systems):

- Task1, Task2, and Task3 represent three heterogeneous task types with the following mean execution times:

$$m_t = \begin{bmatrix} 2.5 & 2.5 & 10 & 10 \\ 2.5 & 2.5 & 5 & 5 \\ 10 & 10 & 2.5 & 2.5 \end{bmatrix}$$

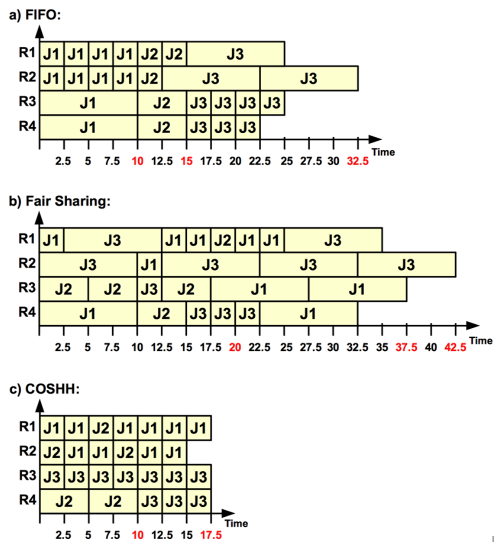
  Here, $m_t(T_i, R_j)$ is the execution time of task T_i on resource R_j.
- Three users submit three jobs to the system, where each job consists of a number of similar tasks. Jobs arrive to the system in the order: Job1, Job2, Job3.
- Users are homogeneous with zero minimum share and a priority equal to one. Each user submits one job to the system as follows:

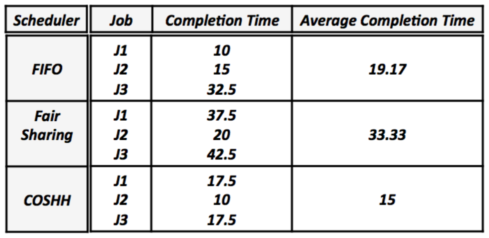
User1:	Job1 (consists of 10 Task1)
User2:	Job3 (consists of 10 Task3)
User3:	Job2 (consists of 5 Task2)

Figure 1 shows the job assignments for FIFO, Fair Sharing, and COSHH schedulers. The completion time of the last task in each job is highlighted to show the overall job completion times. The remainder of this section discusses some of the scheduling challenges in this system.

A. Problem I. Small Jobs Starvation

The FIFO algorithm assigns incoming jobs to the resources based on their arrival times (Figure 1a). Consequently in the FIFO scheduler, execution of the smaller job (Job2) will be delayed significantly. In a heterogeneous Hadoop workload,





Scheduler	Job	Completion Time	Average Completion Time
FIFO	J1 J2 J3	10 15 32.5	19.17
Fair Sharing	J1 J2 J3	37.5 20 42.5	33.33
COSHH	J1 J2 J3	17.5 10 17.5	15

Fig. 1. Job assignment by a)FIFO, b)Fair Sharing, and c)COSHH schedulers, and their average completion times for the heterogeneous Hadoop example.

jobs have different execution times. For such workloads, as the FIFO algorithm does not take into account job sizes, it has the problem that small jobs potentially get stuck behind large ones.

The Fair Sharing and the COSHH algorithms do not have this problem. Fair Sharing puts the jobs in different pools based on their sizes, and assigns a fair share to each pool. As a result, the Fair Sharing algorithm executes different size jobs in parallel. The COSHH algorithm assigns the jobs to the resources based on the job sizes and the execution rates of resources. As a result, it can avoid this problem.

B. Problem II. Sticky Slots

Figure 1b shows the job-resource assignment for the Fair Sharing algorithm. As the users are homogeneous, the Fair Sharing scheduler goes through all of the users' pools, and assigns a slot to one user at each heartbeat. Upon completion of a task, the free slot is assigned to a new task of the same user to preserve fairness among users.

Resource 2 is an inefficient choice for Job3 with respect to the completion time, but the Fair Sharing scheduler assigns this job to this resource multiple times. There is a similar problem for Job1 assigned to resources 3 and 4. Consequently, the average completion times will be increased.

This problem arises when the scheduler assigns a job to the same resource at each heartbeat. The problem is first mentioned in [5] for the Fair Sharing algorithm, where the authors considered the effect of this problem on locality. However, our example shows Sticky Slots can also significantly increase the average completion times, when an inefficient resource is selected for a job.

The FIFO algorithm does not have this problem because it only considers the arrival times in making scheduling decisions. The COSHH algorithm has two levels of classification, which avoids the Sticky Slot problem. Moreover, as the COSHH algorithm matches the job classes to resources with respect to heterogeneity, even if the Sticky Slot issue arises for this algorithm, it will assign a job class to an appropriate resource.

C. Problem III. Resource and Job Mismatch

In a heterogeneous Hadoop system, resources can have different features with respect to their computation or storage units. Moreover, jobs in a heterogeneous workload have different requirements. To reduce the average completion time, it is critical to assign the jobs to resources by considering resource features and job requirements.

The FIFO and the Fair Sharing algorithms both have the problem of resource and job mismatch, as they do not consider heterogeneity in the scheduling. On the other hand, Figure 1c shows that the COSHH algorithm achieves the minimum average completion time (compared to the other algorithms) by applying the following process.

The COSHH algorithm classifies the jobs into three classes: Class1, Class2, and Class3, which contain Job1, Job2, and Job3, respectively. This scheduler solves a Linear Programming problem (LP) to find the best set of suggested job classes for each resource, as follows.

Resource1:	{Class1, Class2}
Resource2:	{Class1, Class2}
Resource3:	{Class2, Class3}
Resource4:	{Class2, Class3}

After computing the suggested sets, the COSHH scheduler considers fairness and minimum share satisfaction to assign a job to a resource. Although the COSHH algorithm assigns Job1 exclusively to Resource1 (Sticky Slot Problem), it does not increase the completion time of Job1. The reason is that COSHH considers the execution times of jobs on resources in selecting Resource1 for Job1. This is one of the main advantages of the COSHH algorithm over FIFO and Fair-Sharing in a heterogeneous system.

IV. Analysis

This section analyzes the performance of the schedulers for a real Hadoop workload in a scalable and heterogeneous environment. The result of this analysis is the justification of a hybrid approach to scheduling. Our experiments include two heterogeneous Hadoop systems: one with a varying number of jobs and one with a varying number of resources.

A. Experimental Environment

The processed workloads include Yahoo! production Hadoop MapReduce traces [8]. The trace is from a cluster at Yahoo!, covering three weeks in late February/early March 2009. It contains a list of job submission and completion times, data sizes of the input, shuffle and output stages, and the running time for the map and reduce functions. Moreover, [8] performs an analysis of the trace to provide classes of "common jobs" using k-means clustering. Table I shows the characteristics of these workloads. MRSIM [12] is used as the MapReduce simulator in these experiments.

It should be clarified that we define the size of jobs based on their execution times reported in [8]. In the experiments, by "small jobs" and "large jobs" we mean the "Small jobs" and "Large data summary" classes in the Yahoo! workload. The default number of jobs is 100, which is sufficient to contain a variety of the behaviours in our Hadoop workload. There are eight users in the experiments with zero minimum shares, and priorities equal to one. The Hadoop block size is set to 128MB, which is the default size in Hadoop 0.21. We set the data replication number to three in all algorithms.

B. Case Study 1: Job Number Scalability

A heterogeneous cluster of six resources (Table II) is used, where the bandwidth between resources is 100 Mbps.

Resources	Slot		Memory	
	$slot\#$	$execRate$	$Capacity$	$RetriveRate$
R_1	2	$5MHz$	$4TB$	$9Gbps$
R_2	16	$500MHz$	$400KB$	$40Kbps$
R_3	16	$500MHz$	$4TB$	$9Gbps$
R_4	2	$5MHz$	$4TB$	$9Gbps$
R_5	16	$500MHz$	$400KB$	$40Kbps$
R_6	2	$5MHz$	$400KB$	$40Kbps$

TABLE II
EXPERIMENTAL RESOURCES

First, to measure the performance of the schedulers in an under-loaded system, a Hadoop system with 5 jobs in its workload was evaluated. Then, we ran multiple experiments by increasing the total job number in the workload to investigate how performance scales with the number of jobs.

Figure 2(i) shows the average completion times for these experiments. Based on the results, when there is a small number of jobs in the workload, and the system is very lightly loaded, the COSHH algorithm has the highest average completion time. This trend continues until the number of submitted jobs reaches the total number of slots in the system (there are 31 map slots and 23 reduce slots on all six resources). After there are around 30 jobs in the workload, the system load reaches the point where all of the submitted jobs can not receive their required slots in the first scheduling round. Therefore, they must wait until a slot becomes available. From this point on, the improvement in average completion time for the COSHH algorithm overcomes its scheduling overhead, and its average completion time is better than the other algorithms. Moreover, at around 30 jobs, the largest size jobs enter the system, which leads to a considerable increase in the average completion times for all of the schedulers.

The average completion time for the Fair Sharing algorithm is initially low. However, once the load in the system increases, and submitted jobs need to be assigned to resources at different heartbeats, its average completion time increases. This is

Job Categories	Duration (sec)	Job	Input	Shuffle	Output	Map Time	Reduce Time
Small jobs	60	114	174 MB	$73MB$	$6MB$	412	740
Fast aggregate	2100	23	568 GB	$76GB$	$3.9GB$	270376	589385
Expand and aggregate	2400	10	206 GB	$1.5TB$	$133MB$	983998	1425941
Transform expand	9300	5	806 GB	$235GB$	$10TB$	257567	979181
Data summary	13500	7	4.9 TB	$78GB$	$775MB$	4481926	1663358
Large data summary	30900	4	31 TB	$937GB$	$475MB$	33606055	31884004
Data transform	3600	36	36 GB	$15GB$	$4.0GB$	15021	13614
Large data transform	16800	1	5.5 TB	$10TB$	$2.5TB$	7729409	8305880

TABLE I
JOB CATEGORIES IN YAHOO! TRACE. MAP TIME AND REDUCE TIME ARE IN TASK-SECONDS, E.G., 2 TASKS OF 10 SECONDS EACH IS 20 TASK-SECONDS [8].

because at each heartbeat, the Fair Sharing algorithm needs to perform sorting and searching over a large sort and search space. Moreover, the Sticky Slot problem and Resource and Job Mismatch (caused by neglecting system heterogeneity) leads to a higher average completion time for the Fair Sharing scheduler compared to the COSHH scheduler. In the case of the FIFO scheduler, there are two factors degrading its performance: Small Job Starvation and the larger ratio of small jobs in the workload. When the large jobs enter the system (between 20 and 30 total jobs), the average completion times significantly increase. However, when there are multiple small jobs in the workload, (between 40 and 50 total jobs), the average completion time decreases.

Figure 2(ii) shows the scheduling time for this experiment. The overheads of all algorithms increase as the number of submitted jobs increases. The growth rate for the COSHH algorithm is higher than the others as a result of its more complicated scheduling process. However, its growth rate decreases as the number of jobs increases. Generally the jobs in Hadoop workloads exhibit some periodic behaviour. The first submitted jobs of a job class can cause a longer classification process. However, because subsequent jobs of the same job class do not need new classes to be defined, the classification process of the COSHH algorithm leads to reduced overheads.

Figure 2(iii) shows the fairness for these experiments. Comparing the algorithms, the Fair Sharing algorithm has the best fairness. This is as expected, because the main goal of this algorithm is improving the fairness metric. The COSHH algorithm has a competitive fairness with the Fair Sharing algorithm.

C. Case Study 2: Resource Number Scalability

This case study evaluates performance when the number of resources varies. The workload in the experiments consists of 100 jobs from the Yahoo! traces in [8]. To define different size clusters, the six types of resources are used, as presented in Table III. The initial experiment was started with six resources, one from each type. For each succeeding experiment, one resource was added to reach 102 resource for the final experiment (i.e. 17 resources of each type).

Figure 2(iv) shows the average completion times for these experiments. Increasing the number of resources reduces the load in the system and improves the average completion time for all of the schedulers. However, this can reduce the chance of local execution for the jobs, which tends to increase the average completion time. Therefore, by increasing the number of resources, first the average completion time of the schedulers reduces until the number of resources reaches approximately 54. Beyond this point, the average completion times increase slightly, because of the locality issue. Similar to the first case study, the Sticky Slot and Small Job Starvation problems affect the performance of the Fair Sharing and FIFO schedulers, respectively. Moreover, they both suffer from the Resource and Job Mismatch problem, increasing their average completion times.

Resources	Slot		Memory	
	$slot\#$	$execRate$	$Capacity$	$RetriveRate$
R_1	2	$5MHz$	$4TB$	$9Gbps$
R_2	2	$500GHz$	$400KB$	$40Kbps$
R_3	2	$500GHz$	$4TB$	$9Gbps$
R_4	2	$5MHz$	$4TB$	$9Gbps$
R_5	2	$500GHz$	$400KB$	$40Kbps$
R_6	2	$5MHz$	$400KB$	$40Kbps$

TABLE III
RESOURCE TYPES

Figure 2(v) shows the scheduling times for the different schedulers. The overheads of the COSHH and the Fair Sharing algorithms get larger as the number of resources increases. The reason is the longer search and sort times in these algorithms. Moreover, the larger number of resources leads to an increase in the classification and LP solving times in the COSHH algorithm. The rate of increase in the COSHH algorithm is higher than the Fair Sharing algorithm. However, its growth rate decreases as the number of jobs increases.

Figure 2(vi) presents the fairness for these experiments. As the number of resources scales up, the fairness metric improves in all algorithms. However, the Fair Sharing algorithm achieves better performance in terms of fairness.

V. HYBRID SOLUTION

Based on the experimental results and analysis, we propose a hybrid scheduler for scalable and heterogeneous Hadoop systems (Figure 3). This scheduler is a combination of the three analyzed algorithms. The selector chooses an appropriate scheduler as the number of jobs and resources scale up or down. However, the overall solution is to use the COSHH algorithm when the system is overloaded (e.g., during peak hours), the FIFO algorithm for underloaded systems (e.g., after hours), and the Fair Sharing algorithm when the system load is balanced.

When the system is underloaded, and the number of free slots is greater than the number of waiting tasks, the scheduler switches to the FIFO algorithm. This can happen when the system has just started or during low load periods. Here, the simple FIFO algorithm can improve the average completion

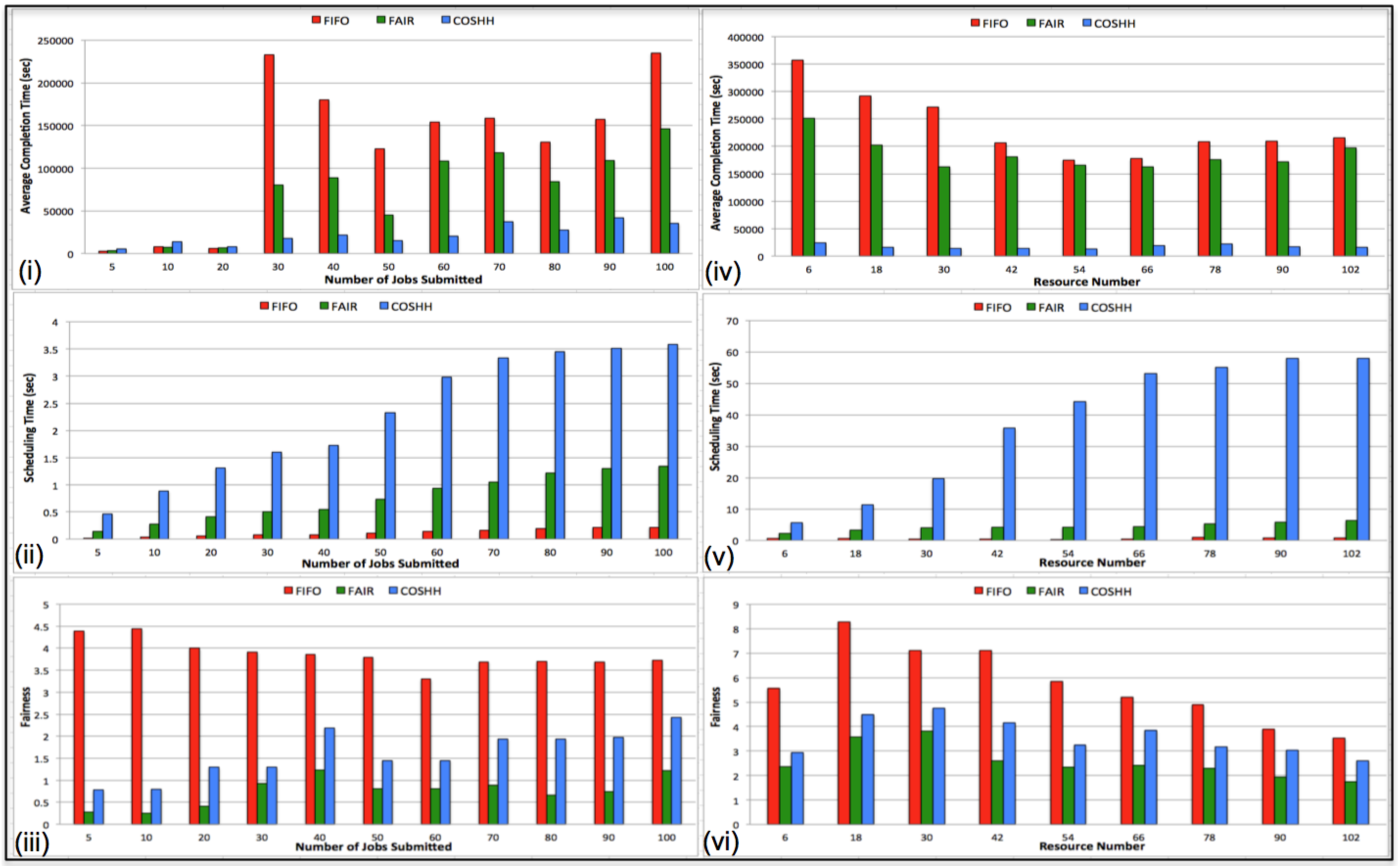
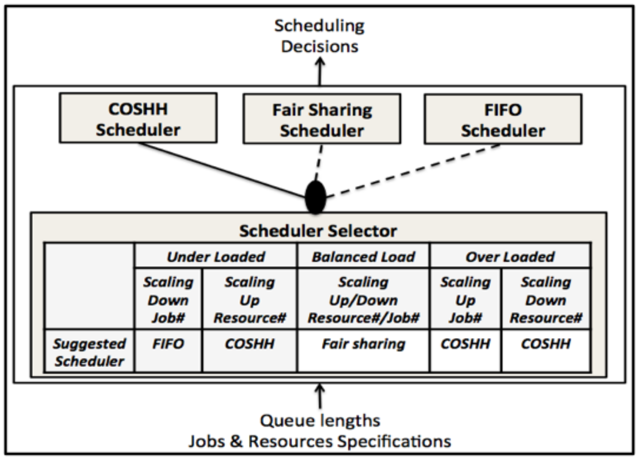


Fig. 2. Performance Metrics for Yahoo! Workload. (i)-(iii): Scaling by Number of Jobs, (iv)-(vi): Scaling by Number of Resources

time with minimum scheduling overhead. However, as the system load increases such that the available number of slots is less than the number of waiting tasks, the hybrid scheduler selects the Fair Sharing algorithm. A good example of this case is when the system has warmed up after starting, and the workload has not yet peaked. In this case, the FIFO algorithm may degrade the system performance with respect to both the average completion time and the fairness metrics. Moreover, as the system is not yet overloaded, using the complex COSHH algorithm can yield large overhead in terms of scheduling time and fairness. When the load increases such that the system is overloaded, and the number of waiting tasks in job queues is quickly increasing, the Fair Sharing algorithm can greatly increase the average completion time. Therefore, the scheduler switches to the COSHH algorithm which improves the average completion time, while avoiding considerable degradation in the fairness metric. The selector needs to define two thresholds to determine the status of the system. Thresholds can be defined based on the system load. One possibility is to use the total number of slots, and the total number of tasks waiting in the scheduling queue, to define the threshold. We expect that other practical guidelines could be developed for this purpose, but leave this for future work.

To evaluate the proposed hybrid solution scheduler, we used another real Hadoop workload, presented in Table IV.



	Under Loaded		Balanced Load	Over Loaded	
	Scaling Down Job#	Scaling Up Resource#	Scaling Up/Down Resource#/Job#	Scaling Up Job#	Scaling Down Resource#
Suggested Scheduler	FIFO	COSHH	Fair sharing	COSHH	COSHH

Fig. 3. Suggested Hybrid Scheduler

The workload contains 100 jobs of a trace from a cluster at Facebook, spanning six months from May to October 2009. In the evaluation, there are 10 users with zero minimum shares, and similar priorities. Each user submits jobs from one category in Table IV. The experimental environment is defined similar to that in Section V.

Figure 4(i)-(iii) shows the average completion time, the scheduling overhead, and fairness as the number of jobs scales.

Job Categories	Duration (sec)	Job	Input	Shuffle	Output	Map Time	Reduce Time
Small jobs	32	126	$21KB$	0	$871KB$	20	0
Fast data load	1260	25	$381KB$	0	$1.9GB$	6079	0
Slow data load	6600	3	10 KB	0	$4.2GB$	26321	0
Large data load	4200	10	405 KB	0	$447GB$	66657	0
Huge data load	18300	3	446 KB	0	$1.1TB$	125662	0
Fast aggregate	900	10	230 GB	$8.8GB$	$491MB$	104338	66760
Aggregate and expand	1800	6	1.9 TB	$502MB$	$2.6GB$	348942	76736
Expand and aggregate	5100	2	418 GB	$2.5TB$	$45GB$	1076089	974395
Data transform	2100	14	255 GB	$788GB$	$1.6GB$	384562	338050
Data summary	3300	1	7.6 TB	$51GB$	$104KB$	4843452	853911

TABLE IV
JOB CATEGORIES IN FACEBOOK TRACE. MAP TIME AND REDUCE TIME ARE IN TASK-SECONDS [8].

These results confirm our observations for the Yahoo! workloads. The hybrid scheduler uses the FIFO, the Fair Sharing, and the COSHH algorithm when the system is underloaded, balanced, and overloaded, respectively. When the number of jobs is scaling up, the first switch between schedulers happens at around 25 jobs in the workload. This is related to the number of tasks waiting and the total slots (31 map slots and 23 reduce slots) on all six resources. The second switch between schedulers is when the number of jobs in the workload is at around 45, which is determined based on the maximum number of tasks in all users' queues of the Fair Sharing algorithm.

Figure 2(iv)-(vi) shows the average completion time, the scheduling time, and fairness, when the number of resources in the system is varying. In these experiments, one switch between schedulers happens when the number of resources is at around 70. This is where the system moves from overloaded to balanced due to the increase in the number of resources.

Finally, it should be noted here that there is a transition period to observe the improved performance after each switch. The transition happens due to the tasks that have already been scheduled or are running as a result of the previous algorithm. The resources first need to complete their jobs assigned by the previous algorithm to be available for jobs scheduled by the newly selected algorithm. In these experiments, the transition happens between 30 and 60 jobs.

VI. RELATED WORK

A job scheduler is an essential component of every Hadoop system. Hadoop uses FIFO as its default scheduler. Some of the performance issues of FIFO were discussed in Section II. The additional schedulers are introduced in [12], where they are collectively known as Fair Sharing. The Fair Sharing algorithm does not achieve good performance with respect to data locality [5]. Delay Scheduler [5] is a complementary algorithm for Fair Sharing which improves the data locality. However, even Delay Scheduler does not consider heterogeneity in the system.

MapReduce was initially designed for small clusters in which a simple and fast scheduling algorithm like FIFO can achieve acceptable performance levels. However, experimental results show that using simple schedulers which do not consider system parameters can cause severe performance degradation in large systems; particularly in systems that share data among multiple users [12]. The next generation of schedulers in Hadoop were introduced by Hadoop on Demand (HOD) [10], which set up private Hadoop clusters on demand for users. HOD allows users to share a common file system while owning private Hadoop clusters on their allocated nodes. This approach failed in practice because it violated the data locality design of the original MapReduce scheduler, and it resulted in poor system utilization.

There are a number of Hadoop schedulers developed for restricted scalable and heterogeneous systems such as Dynamic Priority (DP) [7] and Dominant Resource Fairness (DRF) [11]. The former is a parallel task scheduler which enables users to interactively control their allocated capacity by dynamically adjusting their budgets. The latter addresses the problem of fair allocation of multiple types of resources to users with heterogeneous demands. Finally COSHH [6] is specifically proposed for heterogeneous environments.

This paper evaluates FIFO, Fair Sharing, and COSHH to propose a hybrid solution. Although COSHH has shown promising results for systems with various types of jobs and resources, its scheduling overhead can be a barrier for small and under-loaded systems. This was one of the motivations behind composing different schedulers for a scalable and heterogeneous system. DP was developed for user-interactive environments, differing from our target systems. Similarly, DRF was initially considered to be used instead of COSHH, but this algorithm just considers heterogeneity in the user demands while ignoring resource heterogeneity.

VII. CONCLUSION AND FUTURE WORK

This paper introduces a hybrid scheduler for scalable and heterogeneous Hadoop systems. Performance issues for Hadoop schedulers are analyzed and evaluated for heterogeneous and scalable Hadoop systems. These results suggested a combination of the FIFO, Fair Sharing, and COSHH schedulers is effective, where the selection is based on the load on the system and available system resources.

We plan to extend this work in three directions: (i) the required thresholds specifying system load status will be further investigated. The outcome will be a selection function that considers system parameters including type, number, and complexity of jobs as well as specification of available resources; (ii) the hybrid scheduler will be extended to work for homogeneous environments. The future hybrid scheduler will be smart enough to recognize the degree of heterogeneity in a system and then select the best scheduler for a heterogeneous or homogeneous environment; (iii) the hybrid scheduler has a potential to consider other performance metrics as well. The

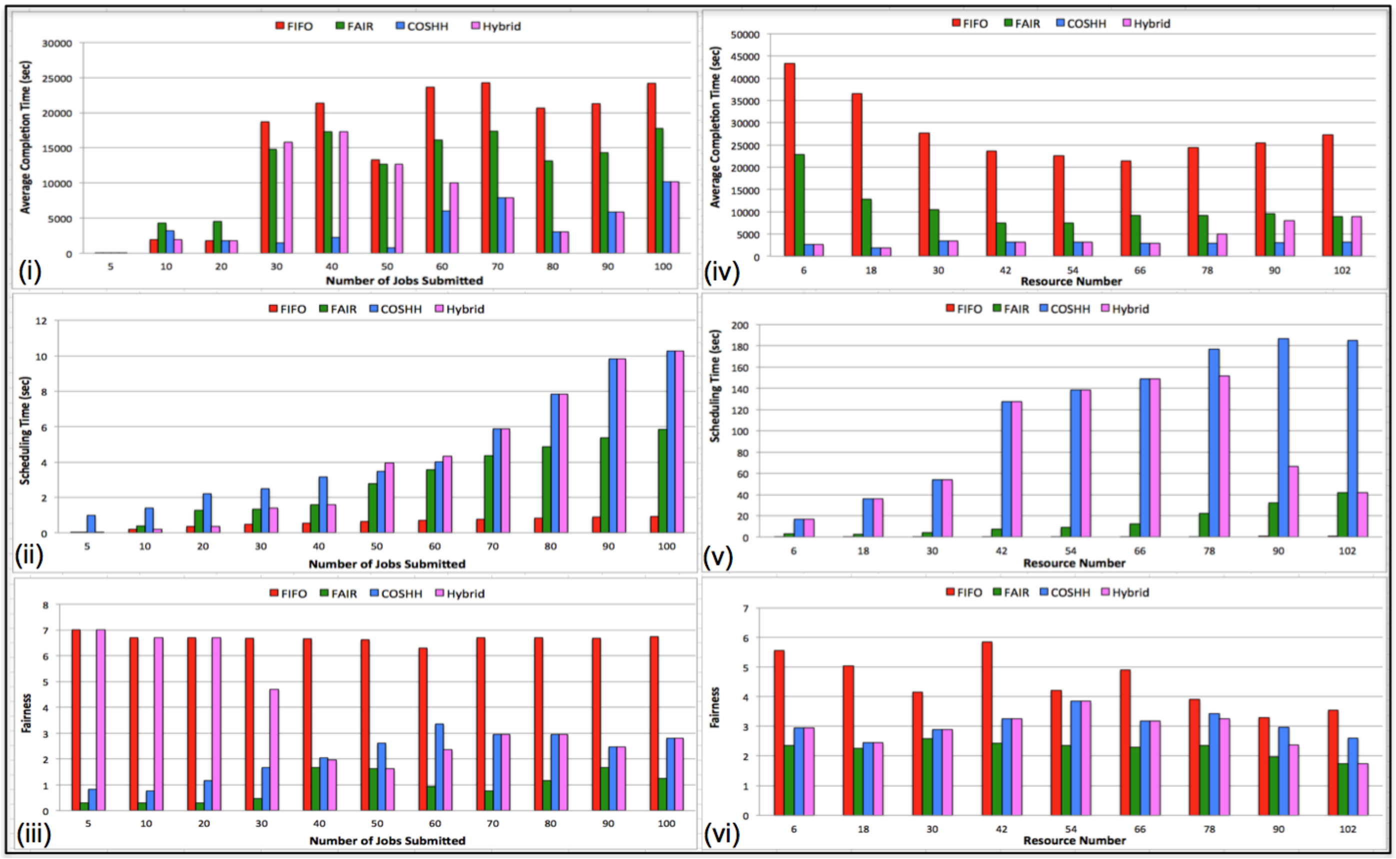


Fig. 4. Performance Metrics for Facebook Workload. (i)-(iii): Scaling by Number of Jobs, (iv)-(vi): Scaling by Number of Resources

scheduler will be extended to receive a desired performance metric as an input, and select an appropriate algorithm with respect to the corresponding metric.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada. A major part of this work was done while both authors were visiting UC Berkeley. In particular, the first author would like to thank Randy Katz, Ion Stoica, Yanpei Chen and Sameer Agarwal for their comments on our research. Also, the authors gratefully acknowledge Facebook and Yahoo! for permission to use their workload traces in this research.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of Cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[2] I. Stoica, "A Berkeley view of Big Data: Algorithms, machines and people," in *UC Berkeley EECS Annual Research Symposium*, 2011.

[3] Apache Hadoop, http://hadoop.apache.org.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107–113, January 2008.

[5] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*, Paris, France, 2010, pp. 265–278.

[6] A. Rasooli and D. G. Down, "An adaptive scheduling algorithm for dynamic heterogeneous hadoop systems," in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '11. Toronto, Ontario, Canada: IBM Corp., 2011, pp. 30–44. [Online]. Available: http://dl.acm.org/citation.cfm?id=2093889.2093893

[7] T. Sandholm and K. Lai, "Dynamic proportional share scheduling in Hadoop," in *Proceedings of the 15th Workshop on Job Scheduling Strategies for Parallel Processing*. Heidelberg, 2010, pp. 110–131.

[8] Y. Chen, A. Ganapathi, R. Griffith, and R. H. Katz, "The case for evaluating MapReduce performance using workload suites," in *Proceedings of the 19th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2011, pp. 390–399.

[9] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads," *Proceedings of the international conference on Very Large Data Bases (VLDB) Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2367502.2367519

[10] Apache, "Hadoop On Demand documentation," 2007, [Online; accessed 30-November-2010]. [Online]. Available: http://hadoop.apache.org/common/docs/r0.17.2/hod.html

[11] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2011, pp. 24–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972490

[12] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user MapReduce clusters," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, April 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-55.html