COSHH: A Classification and Optimization based Scheduler for Heterogeneous Hadoop Systems

Aysan Rasooli[a], Douglas G. Down[a]

[a]*Department of Computing and Software, McMaster University, L8S 4K1, Canada*

Abstract

A Hadoop system provides execution and multiplexing of many tasks in a common datacenter. There is a rising demand for sharing Hadoop clusters amongst various users, which leads to increasing system heterogeneity. However, heterogeneity is a neglected issue in most Hadoop schedulers. In this work we design and implement a new Hadoop scheduling system, named COSHH, which considers heterogeneity at both the application and cluster levels. The main objective of COSHH is to improve the mean completion time of jobs. However, as it is concerned with other key Hadoop performance metrics, our proposed scheduler also achieves competitive performance under minimum share satisfaction, fairness and locality metrics with respect to other well-known Hadoop schedulers.

Keywords: Hadoop System, Scheduling System, Heterogeneous Hadoop

1. Introduction

Hadoop systems were initially designed to optimize the performance of large batch jobs such as web index construction [1]. However, due to its advantages, the number of applications running on Hadoop is increasing, which leads to a growing demand for sharing Hadoop clusters amongst multiple users [1]. Various types of applications submitted by different users require the consideration of new aspects in designing a scheduling system for Hadoop. One of the most important aspects which should be considered is heterogeneity in the system. Heterogeneity can be at both the application and the cluster levels. Application level heterogeneity is taken into account in some recent research on Hadoop schedulers [2]. However, to the best of our knowledge, cluster level heterogeneity is a neglected aspect in designing Hadoop schedulers. In this work, we introduce a new scheduling system (called COSHH) designed and implemented for Hadoop, which considers heterogeneity at both application and cluster levels.

The main approach in our scheduling system is to use system information to make better scheduling decisions, which leads to improving the performance. COSHH consists of several components. The component which gathers system information was first introduced in [3], and is further developed in [4], which provides a means to estimate the mean job execution time based on the structure of the job, and the number of map and reduce tasks in each job. The main motivations for our scheduler are as follows:

- **Scheduling based on fairness, minimum share requirements, and the heterogeneity of jobs and resources.** In a Hadoop system, satisfying the minimum shares of users is the first critical issue. The next important issue is fairness. We design a scheduling algorithm which has two stages. In the first stage, the algorithm considers the satisfaction of the minimum share requirements for all users. Then, in the second stage, the algorithm considers fairness for all users. Most current Hadoop scheduling algorithms consider fairness and minimum share objectives without considering heterogeneity of the jobs and the resources. One of the advantages of COSHH is that while it addresses the fairness and the minimum share requirements, it does this in a way that makes efficient assignments, by considering the heterogeneity in the system. The system heterogeneity is defined based on job requirements (e.g., estimated execution time) and resource features (e.g., execution rate). Consequently, the proposed scheduler reduces the average completion time of the jobs.

- **Reducing the communication cost in the Hadoop system.** The Hadoop system distributes tasks among the resources to reduce a job's completion time. However, Hadoop does not consider communication costs. In a large cluster with heterogenous resources, maximizing a task's distribution may result in overwhelmingly large communication overhead. As a result, a job's completion time will be increased. COSHH considers the heterogeneity and distribution of resources in the task assignment.

- **Reducing the search overhead for matching jobs and resources.** To find the best matching of jobs and resources in a heterogeneous Hadoop system, an exhaustive search is required. COSHH uses classification and optimization techniques to restrict the search space. Jobs are categorized based on their requirements. Every time a resource is available, it searches through the classes instead of the individual jobs to find the best matching (using optimization techniques). The solution of the optimization problem results in the set of suggested classes for each resource, used for making routing decisions. Moreover, to avoid adding significant overhead, COSHH limits the number of times that classification and optimization are performed in the scheduler.

- **Increasing locality.** In order to increase locality, we should increase the probability that tasks are assigned to resources which also store their input data. COSHH makes a scheduling decision based on the suggested set of job classes for each resource. Therefore, the required data of the suggested classes of a resource can be replicated on that resource. This can lead to increased locality, in particular in large Hadoop clusters, where locality is more critical.

We use a Hadoop simulator, MRSIM [5], and extend it to evaluate our proposed scheduler. The four most common performance metrics for Hadoop systems: locality, fairness, minimum share satisfaction, and average completion time, are implemented. The performance of COSHH is compared with two commonly used Hadoop scheduling algorithms, the FIFO algorithm and the Fair-sharing algorithm [1]. The results show that COSHH has significantly better performance in reducing the average completion time, and satisfying the required minimum shares. Moreover, its performance for the locality and the fairness performance metrics is very competitive with the other two schedulers. Furthermore, we demonstrate the scalability of COSHH based on the number of jobs and resources in the Hadoop system. The sensitivity of the proposed algorithm to errors in the estimated job execution times is examined. Our results show that even in a system with as much as 40% error in estimating job execution times, the COSHH algorithm significantly improves average completion times.

To evaluate the overhead of the COSHH scheduler, we present its scheduling time, and compare it with the other schedulers. The improvement in average completion time is achieved at the cost of increasing the overhead of scheduling. However, the additional overhead for the COSHH algorithm, compared to the improvement for average completion time, is in most cases negligible.

The remainder of this paper is organized as follows. The high level architecture of COSHH is introduced in Section 2. Details of the two main components in our proposed scheduler are presented in Sections 3 and 4. In Section 5, we present details of the evaluation environment, and study the performance of COSHH with two well-known real Hadoop workloads. Section 6 provides further discussion about the performance of the COSHH scheduler, and analyzes it from sensitivity and scalability perspectives. Implementation on an actual cluster is presented in Section 7. Current Hadoop scheduling algorithms are given in Section 8. We discuss future directions in the concluding section.

2. Proposed Hadoop Scheduling System

The high level architecture of COSHH is presented in Figure 1. In this section we present a brief overview of all of the components. We will provide greater detail for the main components in the next two sections.

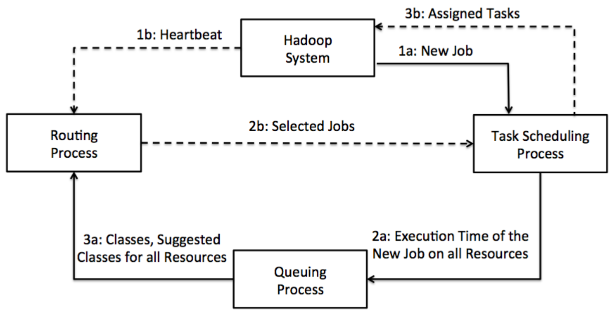

Figure 1: The high level architecture of COSHH

A typical Hadoop scheduler receives two main messages from the Hadoop system: a message signalling a new job arrival from a user, and a heartbeat message from a free resource. Therefore, COSHH consists of two main processes, where each process is triggered by receiving one of these messages. Upon receiving a new job, the scheduler performs the queuing process to store the incoming job in an appropriate queue. Upon receiving a heartbeat message, the scheduler triggers the routing process to assign a job to the current free resource. In Figure 1, the flows of the job arrival and heartbeat messages are presented by solid and dashed lines, respectively.

The high level architecture of COSHH consists of four components: the Hadoop system, the task scheduling process, the queuing process, and the routing process. The task scheduling process estimates the execution time of an incoming job on all resources. These estimates are passed to the queuing process to choose an appropriate queue for the incoming job. The routing process selects a job for the available free resource, and sends it to the task scheduling process. Using the selected job's characteristics, the task scheduling process assigns tasks of the selected job to available slots of the free resource. The Hadoop system and task scheduling process are introduced in this section, and the detailed description of the queuing and routing processes are provided in Sections 3 and 4, respectively.

2.1. Hadoop System

The Hadoop system consists of a cluster, which is a group of linked resources. The data in the Hadoop system is organized into files. The users submit jobs to the system, where each job consists of a number of tasks. Each task is either a *map task* or a *reduce task*. The Hadoop components related to our research are described as follows:

1. The cluster consists of a set of resources, where each resource has a computation unit and a data storage unit. The computation unit consists of a set of slots (in most Hadoop systems, each CPU core is considered as one slot) and the data storage unit has a specific capacity. We assume a cluster with M resources as follows:

$$Cluster = \{R_1, \ldots, R_M\}$$
$$R_j =< Slots_j, Mem_j >$$

   - $Slots_j$ is the set of slots in resource R_j, where each slot ($slot_j^k$) has a specific execution rate ($exec\_rate_j^k$). Generally, slots belonging to one resource have the same execution rate. A resource R_j has the following set of s slots:

   $$Slots_j = \{slot_j^1, \ldots, slot_j^s\}$$

   - Mem_j is the storage unit of resource R_j, which has a specific capacity ($capacity_j$) and data retrieval rate ($retrieval\_rate_j$). The data retrieval rate of resource R_j depends on the bandwidth within the storage unit of this resource.

2. 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). We assume that there are f files in the system, and each file is divided into l slices, which are defined as follows:

$$Files = \{F_1, \ldots, F_f\}$$
$$F_i = \{slice_i^1, \ldots, slice_i^l\}$$

3. We assume that there are N users in the Hadoop system, where each user (U_i) submits a set of jobs to the system ($Jobs_i$) as follows:

$$Users = \{U_1, \ldots, U_N\}$$
$$U_i =< Jobs_i >$$
$$Jobs_i = \{J_i^1, \ldots, J_i^n\},$$

where J_i^d denotes job d submitted by user U_i, and n is the total number of jobs submitted by this user. The Hadoop system assigns a priority and a minimum share to each user based on a particular policy (e.g. the pricing policy of [6]).
The priority is an integer which shows the relative importance of a user. Based on the priority ($priority_i$) of a user U_i, we define a corresponding weight ($weight_i$), where the weight can be any integer or fractional number. The number of slots assigned to user U_i depends on her weight ($weight_i$). The minimum share of a user U_i (min_share_i) is the minimum number of slots that the system must provide for user U_i at each point in time.
In a Hadoop system, the set of submitted jobs of a user is dynamic, meaning that the set of submitted jobs for user U_i at time t_1 may be completely different at time t_2. Each job (J_i) in the system consists of a number of *map task*s and *reduce task*s. A job J_i is represented by

$$J_i = Maps_i \cup Reds_i,$$

where $Maps_i$ and $Reds_i$ are the sets of *map task*s and *reduce task*s of this job, respectively. The set $Maps_i$ of job J_i is denoted by

$$Maps_i = \{MT_i^1, \ldots, MT_i^{m'}\}.$$

Here, m' is the total number of map tasks, and MT_i^k is *map task k* of job J_i. Each map task MT_i^k performs some processing on the slice ($slice_j^l \in F_j$) where the required data for this task is located.
The set $Reds_i$ of job J_i is denoted by

$$Reds_i = \{RT_i^1, \ldots, RT_i^{r'}\}.$$

Here, r' is the total number of reduce tasks, and RT_i^k is *reduce task k* of job J_i. Each reduce task RT_i^k receives and processes the results of some of the *map task*s of job J_i.
The value $mean\_execTime(J_i, R_j)$ defines the mean execution time of job J_i on resource R_j, and the corresponding execution rate is defined as follows:

$$mean\_execRate(J_i, R_j) =$$
$$1/mean\_execTime(J_i, R_j).$$

2.2. Task Scheduling Process

Upon a new job arrival, an estimate of its mean execution times on the resources is required. The task scheduling process component uses a task duration predictor to estimate the mean execution times of the incoming job on all resources ($mean\_execTime(J_i, R_j)$). This component is a result of research in the AMP lab at UC Berkeley [4].

To define the prediction algorithm, first various analyses are performed in [4], to identify important log signals. Then, the prediction algorithm is introduced using these log signals, and finally the accuracy of the prediction algorithm is evaluated on real Hadoop workloads. The prediction algorithm should be able to make a decision within a matter of microseconds, with fairly high accuracy. To achieve this goal, the estimator consists of two parts: the first part, *chrond*, refers to a daemon running in the background. It is responsible for analyzing Hadoop history log files as well as monitoring cluster utilizations every specified time interval. For example, for Facebook and Yahoo! workloads, an interval of every six hours is able to provide the desired accuracy [7].

Periodically, k-means clustering [8] is applied on this data, which keeps track of the cluster boundaries. On the other hand, the second part, *Predictor*, is an on-the-spot decision engine. Whenever a new job arrives, the *Predictor* classifies its tasks into various categories depending on the file they operate on, the total cluster utilization at that point in time, and the input bytes they read, by consulting the lookup table populated by *chrond*. Finally, it returns the mean job execution times.

The refined *Predictor* algorithm for COSHH is provided in [9]. Accuracy experiments for *Chronos* are provided in [4] on 46 GB of Yahoo! and Facebook cluster logs. The results show that around 90% of map tasks are predicted within 80% accuracy. Further, about 80% of reduce tasks are predicted within 80% accuracy. Most importantly, the addition of *Chronos* to the existing Hadoop Schedulers did not result in any significant performance degradation [4].

3. Queuing Process

Figure 2 shows the stages of the queuing process. The two main approaches used in the queuing process are classification and optimization based approaches, introduced in detail in Sections 3.1, and 3.2, respectively. At a high level, when a new job arrives, the classification approach specifies the job class, and stores the job in the corresponding queue. If the job does not fit any of the current classes, the list of classes is updated to add a class for the incoming job. The optimization approach is used to find an appropriate matching of job classes and resources. An optimization problem is defined based on the properties of the job classes and features of the resources. The result of the queuing process, which is sent to the routing process, contains the list of job classes and the suggested set of classes for each resource.

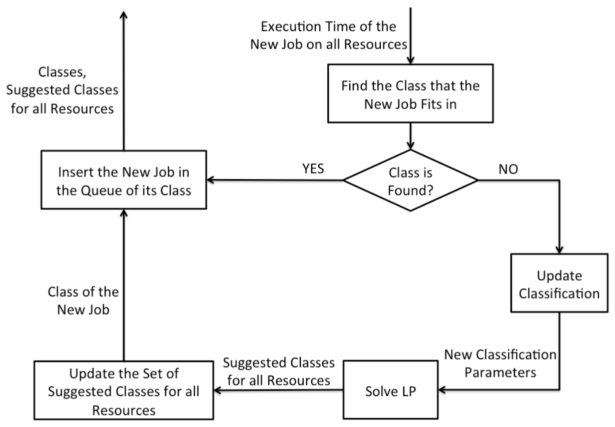
Execution Time of the New Job on all Resources
Find the Class that the New Job Fits in
Class is Found?
YES
NO
Insert the New Job in the Queue of its Class
Classes, Suggested Classes for all Resources
Update Classification
New Classification Parameters
Solve LP
Suggested Classes for all Resources
Update the Set of Suggested Classes for all Resources
Class of the New Job

Figure 2: The Queuing Process

The classification and optimization approaches are used to reduce the search space in finding an appropriate matching of resources and jobs. Moreover, by using these approaches, we consider the heterogeneity in the system, and reduce completion times. However, using these two approaches can add overhead to the scheduling process. In order to avoid significant overhead, we limit the number of times that these steps are performed. Also, we perform the classification and the optimization approaches by methods with small overheads. In the following, we first introduce the details of the classification and optimization approaches, and later we will provide a complete algorithm for the queuing process.

3.1. Classification-based Approach

Investigations on real Hadoop workloads show that it is possible to determine classes of “common jobs” [10]. COSHH uses k-means, a well-known clustering method [8], for classification. This method is used for classifying jobs in real Hadoop workloads [10].

We designed our scheduler based on the fact that there are two critical criteria with different levels of importance in a Hadoop system. The first criterion, imposed by the Hadoop provider, is satisfying the minimum shares. The Hadoop providers guarantee that upon a user's request at any time, her minimum share will be provided immediately (if feasible). The second criterion, important to improve the overall system performance, is fairness. Considering fairness prevents starvation of any user, and divides the resources among the users in a fair manner. Minimum share satisfaction has higher criticality than fairness. Therefore, COSHH has two classifications, to consider these issues for first minimum share satisfaction, then for fairness. In the primary classification (for minimum share satisfaction), only the jobs whose users have $min\_share > 0$ are classified, and in the secondary classification (for fairness) all of the jobs in the system are considered. The jobs whose users have $min\_share > 0$ are considered in both classifications. The reason is that when a user asks for more than her minimum share, first her minimum share is given to her immediately through the primary classification. Then, extra shares should be given to her in a fair way by considering all users through the secondary classification.

In both classifications, jobs are classified based on their features (i.e. priority, mean execution rate on the resources ($mean\_execRate(J_i, R_j)$), and mean arrival rate). The set of classes generated in the primary classification is defined as $JobClasses1$, where an individual class is denoted by C_i. Each class C_i has a given priority, which is equal to the priority of the jobs in this class. The estimated mean arrival rate of the jobs in class C_i is denoted by α_i, and the estimated mean execution rate of the jobs in class C_i on resource R_j is denoted by $\mu_{i,j}$. Hence, the heterogeneity of resources is completely addressed with $\mu_{i,j}$. The total number of classes generated with this classification is assumed to be F, i.e.

$$JobClasses1 = \{C_1, \ldots, C_F\}.$$

The secondary classification generates a set of classes defined as $JobClasses2$. As in the priority classification, each class, denoted by C'_i, has priority equal to the priority of the jobs in this class. The mean arrival rate of the jobs in class C'_i is equal to α'_i, and the mean execution rate of the jobs in class C'_i on resource R_j is denoted by $\mu'_{i,j}$. We assume that the total number of classes generated with this classification is F', i.e.

$$JobClasses2 = \{C'_1, \ldots, C'_{F'}\}.$$

For example, Yahoo! uses the Hadoop system in production for a variety of products (job types) [11]: Data Analytics, Content Optimization, Yahoo! Mail Anti-Spam, Ad Products, and several other applications. Typically,

User	Job Type	min_share	priority
User1	Advertisement Products	50	3
User2	Data Analytics	20	2
User3	Advertisement Targeting	40	3
User4	Search Ranking	30	2
User5	Yahoo! Mail Anti-Spam	0	1
User6	User Interest Prediction	0	2

Table 1: The Hadoop System Example (Exp1)

the Hadoop system defines a user for each job type, and the system assigns a minimum share and a priority to each user. For example, assume a Hadoop system (called Exp1) with the parameters in Table 1. The corresponding jobs at a given time t, are given in Table 2, where the submitted jobs of a user are based on the user's job type (e.g., J_4, submitted by User1, is an advertisement product, while job J_5 is a search ranking job). The primary classification of the

User	Job Queue
User1	$\{J_4, J_{10}, J_{13}, J_{17}\}$
User2	$\{J_1, J_5, J_9, J_{12}, J_{18}\}$
User3	$\{J_2, J_8, J_{20}\}$
User4	$\{J_6, J_{14}, J_{16}, J_{21}\}$
User5	$\{J_7, J_{15}\}$
User6	$\{J_3, J_{11}, J_{19}\}$

Table 2: The job queues in Exp1 at time t

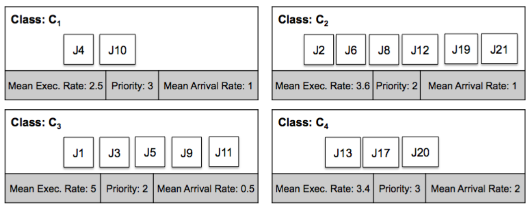

Figure 3: The primary classification of the jobs in Exp1 system at time t

jobs in the Exp1 system, at time t, is presented in Figure 3. Note that here we assume that there is just one resource in the system. The secondary classification of system Exp1, at time t, is shown in Figure 4. The parameter k (used for k-means clustering) is set in our systems to be the number of users. Based on the studies on Facebook and Yahoo! workloads, as well as studies of other Hadoop workloads, jobs sent from a user to a Hadoop cluster can be classified to belong to the same class ([10]). Setting k to a larger number does not have significant impact on performance, as the matching is defined based on heterogeneity of jobs and resources.

3.2. Optimization based Approach

After classifying the incoming jobs, and storing them in their appropriate classes, the scheduler finds a matching of jobs and resources. The optimization approach used in our scheduler first constructs a linear program (LP) which considers properties of the job classes and features of the

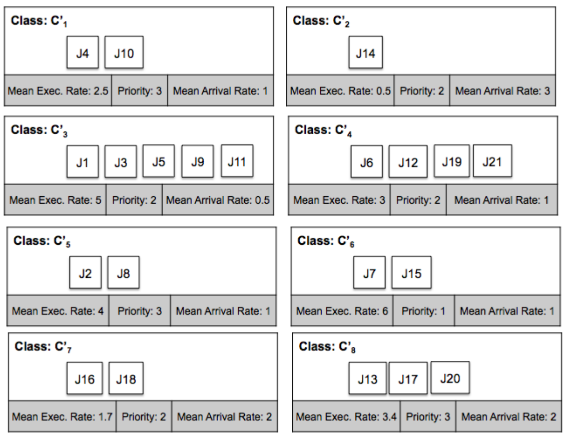

Figure 4: The secondary classification of the jobs in Exp1 system at time t

resources. The scheduler then solves this LP to find a set of suggested classes for each resource.

An LP is defined for each of the classifications. The first LP is defined for classes in the set $JobClasses1$ as follows:

$$\max \lambda$$

$$s.t. \sum_{j=1}^{M} \mu_{i,j} \times \delta_{i,j} \geq \lambda \times \alpha_i, \; for\ all\ i = 1, \ldots, F, \tag{1}$$

$$\sum_{i=1}^{F} \delta_{i,j} \leq 1, \; for\ all\ j = 1, \ldots, M, \tag{2}$$

$$\delta_{i,j} \geq 0, \; for\ all\ i = 1, \ldots, F, \; and\ j = 1, \ldots, M. \tag{3}$$

Here λ is interpreted as the maximum capacity of the system, and $\delta_{i,j}$ is the proportion of resource R_j which is allocated to class C_i. Moreover, M is the total number of resources, and F is the total number of classes generated in the primary classification ($|JobClasses1|$). This optimization problem increases the arrival rates of all the classes by a fixed proportion, to minimize the load in the system, while in (1) the system is kept stable. After solving this LP, we have the allocation matrix δ, whose (i, j) element is $\delta_{i,j}$. Based on the results of this LP, we define the set SC_j for each resource R_j as

$$SC_j = \{C_i : \delta_{i,j} \neq 0\}.$$

For example, consider a system with two classes of jobs, and two resources ($M = 2$, $F = 2$), in which the arrival and execution rates are $\alpha = \begin{bmatrix} 2.45 & 2.45 \end{bmatrix} and$ $\mu = \begin{bmatrix} 9 & 5 \\ 2 & 1 \end{bmatrix}$, respectively. Solving the above LP gives $\lambda = 1.0204$ and $\delta = \begin{bmatrix} 0 & 0.5 \\ 1 & 0.5 \end{bmatrix}$. Therefore, the sets SC_1 and SC_2 for resources R_1 and R_2 will be $\{C_2\}$ and $\{C_1, C_2\}$, respectively. These two sets define the suggested classes for each resource, i.e. upon receiving a heartbeat from resource R_1, a job from class C_2 should be selected. However, upon receiving a heartbeat from resource R_2, either a job from class C_1 or C_2 should be chosen. Even though resource R_1 has the fastest rate for class C_1, the algorithm does not assign any jobs of class C_1 to it. If the system is highly loaded, it turns out that the average completion time of the jobs will decrease if resource R_1 only executes class C_2 jobs.

The second optimization problem is used for the secondary classification. The scheduler defines an LP similar to the previous one, for classes in the set $JobClasses2$. However, in this LP the parameters λ, $\mu_{i,j}$, $\delta_{i,j}$, α_i, and F are replaced by λ', $\mu'_{i,j}$, $\delta'_{i,j}$, α'_i, and F', respectively:

$$\max \lambda'$$

$$s.t. \sum_{j=1}^{M} \mu'_{i,j} \times \delta'_{i,j} \geq \lambda' \times \alpha'_i, \; for\ all\ i = 1, \ldots, F', \tag{4}$$

$$\sum_{i=1}^{F'} \delta'_{i,j} \leq 1, \; for\ all\ j = 1, \ldots, M, \tag{5}$$

$$\delta'_{i,j} \geq 0, \; for\ all\ i = 1, \ldots, F', \; and\ j = 1, \ldots, M. \tag{6}$$

After solving this LP, we will have the matrix δ', whose (i, j) element is $\delta'_{i,j}$. We define the set SC'_j for each resource R_j as the set of classes which are allocated to this resource based on the result of this LP, where $SC'_j = \{C'_i : \delta'_{i,j} \neq 0\}$.

COSHH uses the sets of suggested classes SC_R and SC'_R for both making scheduling decisions and improving locality in the Hadoop system. The scheduling decision is made by the routing process, and locality can be improved by replicating input data on multiple resources in the Hadoop system. Most current Hadoop schedulers randomly choose three resources for replication of each input data [1, 12]. However, COSHH uses the sets of suggested classes, SC_R and SC'_R, to choose replication resources. For each input data, the initial incoming jobs using this data are considered, and from all the suggested resources for these jobs, three of them are randomly selected for storing replicas of the corresponding input data. Since in this work we evaluate our algorithm on a small cluster, we only consider the initial incoming jobs to determine the replication resources. However, in large Hadoop clusters with a high variety of available network bandwidths, developing our proposed replication method to consider the updates caused by later incoming jobs, could lead to significant improvement in the locality. We leave this as future work.

Algorithm 1 Queuing Process

```
When a new Job (say $J$) arrives
Get execution time of $J$ from Task Scheduling Process
  if $J$ fits in any class (say $C_i$) then
    add $J$ to the queue of $C_i$
  else
    use $k$-means clustering to update the job classification
    find a class for $J$ (say $C_j$) , and add $J$ to its queue
    solve optimization problems, and get two sets of suggested
    classes, $SC_R$ and $SC'_R$
  end if
  send $SC_R$, $SC'_R$ and both sets of classes ($JobClasses1$ and
  $JobClasses2$) to the routing process
```

We used the IBM ILOG CPLEX optimizer [13] to solve the LPs. A key feature of this optimizer is its performance in solving very large optimization problems, and the speed required for highly interactive analytical decision support applications [14]. As a result, solving the optimization problems in COSHH does not add considerable overhead.

Now that we have defined the two main approaches of our proposed queuing process, the complete algorithm is presented in Algorithm 1.

4. Routing Process

When the scheduler receives a heartbeat message from a free resource, say R_j, it triggers the routing process. The routing process receives the sets of suggested classes SC_R and SC'_R from the queuing process, and uses them to select a job for the current free resource. This process selects a job for each free slot in the resource R_j, and sends the selected job to the task scheduling process. The task scheduling process chooses a task of the selected job, and assigns the task to its corresponding slot.

Here, it should be noted that the scheduler is not limiting each job to just one resource. When a job is selected, the task scheduling process assigns a number of appropriate tasks of this job to available slots of the current free resource. If the number of available slots is fewer than the number of uncompleted tasks for the selected job, the job will remain in the waiting queue. Therefore, at the next heartbeat message from a free resource, this job is considered in making the scheduling decision; however, tasks already assigned are no longer considered. When all tasks of a job are assigned, the job will be removed from the waiting queue.

Algorithm 2 presents the routing process. There are two stages in this algorithm to select jobs for the available slots of the current free resource. In the first stage, the jobs of classes in SC_R are considered, where the jobs are selected in the order of their minimum share satisfaction. This means that a user who has the highest distance to achieve her minimum share will get a resource share sooner. However, in the second stage, jobs for classes in SC'_R are considered, and jobs are selected in the order defined by the current shares and priorities of their users. In this way, the scheduler addresses fairness amongst the users. In each stage, if there are two users with exactly the same conditions, we randomly choose between them.

It should be noted that COSHH is a dynamic scheduler. Based on any variation in the Hadoop workload and resources, the classification and LP solver components can update the scheduling decisions accordingly.

Algorithm 2 Routing Process

```
When a heartbeat message is received from a resource (say $R$)
$N_{FS}$ = number of free slots in $R$
  while $N_{FS} \neq 0$ and there is a job ($J$) whose
      $user.minShare - user.currentShare > 0$
    and
      $class \in SC_R$
    and
    $((user.minShare - user.currentShare) \times weight)$ is maximum
  do
     add $J$ to the set of selected jobs ($J_{selected}$)
     $N_{FS} = N_{FS} - 1$
  end while
  while $N_{FS} \neq 0$ and there is a job ($J$) whose
      $class \in SC'_R$
    and
    $(user.currentShare/weight)$ is minimum
  do
     add $J$ to the set of selected jobs ($J_{selected}$)
     $N_{FS} = N_{FS} - 1$
  end while
  send the set $J_{selected}$ to the Task Scheduling Process to choose a
  task for each free slot in $R$.
```

5. Experimental Results

In this section we evaluate our proposed scheduling system on real Hadoop workloads. First, we define the performance metrics considered. Later, we introduce our experimental environment and workload, and finally we present the evaluation results.

5.1. Performance Metrics

We define the function $Demand(U,t)$ as the set of unassigned tasks for user U at time t. Also, the function $AssignedSlots(U,t)$ is defined as the set of slots executing tasks from user U at time t. We consider an experiment which is run for time T. Using these definitions, we define four Hadoop performance metrics:

1. $AveragecompletionTime$ 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. A user whose current demand is not zero ($|Demand(U,t)| > 0$), and whose current share is less than her minimum share ($|AssignedSlots(U,t)| < U.min\_share$), has the following $UserDissatisfaction$:

   $UserDissatisfaction(U,t) =$
   $$\frac{U.min\_share - |AssignedSlots(U,t)|}{U.min\_share} \times U.weight,$$
   where $U.weight$ denotes the weight, and $U.min\_share$ is the minimum share of the user U. The total distance of all users from their min_share is defined by $Dissatisfaction(t)$ as follows:

   $Dissatisfaction(t) =$
   $$\sum_{\forall U \in Users} UserDissatisfaction(U,t).$$
   Comparing two algorithms, the algorithm which has smaller $Dissatisfaction(T)$ has better performance.
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 the user's share should be proportional to their weight. In order to compute the fairness of an algorithm, we should take into account the number of slots which are assigned to each user beyond her minimum share, which is computed as $\Delta(U,t) = AssignedSlots(U,t) - U.min\_share$. Then, the average additional share of all users with the same weight ($Users_w$) is defined as:
   $$avg(w,t) = \frac{\sum_{U \in Users_w} \Delta(U,t)}{|Users_w|},$$
   where $Users_w = \{U | U \in Users \wedge U.weight = w\}$. $Fairness(t)$ is computed by the sum of distances of all the users in one weight level from the average amount of that weight level:
   $Fairness(t) =$
   $$\sum_{w \in weights} \sum_{U \in Users_w} |\Delta(U,t) - avg(w,t)|.$$
   Comparing two algorithms, the algorithm which has lower $Fairness(T)$ achieves better performance.
4. *Locality* is defined as the proportion of tasks which are running on the same resource as where their stored data are located. Since in the Hadoop system the input data size is large, and the *map task*s of one job are required to send their results to the *reduce task*s of that job, the communication cost can be quite significant. 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. Comparing two scheduling algorithms, the algorithm which has larger *Locality*(T) has better performance.
5. Finally, in each experiment, we present *Scheduling Overhead* as the time spent for scheduling all of the incoming jobs.

5.2. Experimental Environment

We use MRSIM [5], a MapReduce simulator, to simulate a Hadoop cluster and evaluate our scheduler. This simulator is based on discrete event simulation, and accurately models the Hadoop environment. The simulator on the one hand allows us to measure scalability of the MapReduce based applications easily and quickly, while capturing the effects of different Hadoop configurations.

We extended this simulator to measure the five main Hadoop performance metrics defined above. A job submission process component is added to the MRSIM architecture to be able to submit the desired stream of jobs in our workload to the Hadoop system. The arrival times of the jobs are calculated based on the workload information. The job submitter is triggered based on the arrival times to submit a new job to the system. Also, using this component we can define various users with different minimum shares and priorities. The input data file of each job is directed to the dfs component in MRSIM to calculate the slices and store the file. The HJobTracker component in MRSIM is extended to receive the desired scheduling algorithm name from the configuration file. The scheduling algorithms are implemented inside the scheduler class, and are called from the HJobTracker component in each heartbeat. The COSHH scheduler is called for classifying the incoming job, and also for assigning a new job when there is a heartbeat message from a resource. The COSHH scheduler consists of several classes for the Classification, Routing and LP solving processes. There is also an evaluation part added to the simulator which calculates the desired performance metrics whenever a job is completed. The detailed implementation architecture of COSHH is provided in [9].

Our experimental environment consists of a cluster of six heterogeneous resources. The resources' features are presented in Table 3. The bandwidth between the resources is 100Mbps.

Resources	Slot		Mem	
	slot#	*execRate*	*Capacity*	*RetrieveRate*
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 3: Experimental resources

We used two production Hadoop MapReduce traces, presented in [10]. One trace is from a cluster at Facebook, spanning six months from May to October 2009. The other trace is from a cluster at Yahoo!, covering three weeks in late February/early March 2009. Both traces contain a list of job submission and completion times, data sizes of the input, shuffle and output stages, and the running time of map and reduce functions. The arrival rates of the jobs in our experiments are defined by considering the number of jobs in each Facebook and Yahoo! trace, the total number of submitted jobs in each experiment, and also the total number of classes of jobs in each trace. The proportion of the number of each job class to the total number of jobs in our workload is the same as the actual trace. The research in [10] performs an analysis of both traces, which provides classes of "common jobs" for each of the Facebook and Yahoo! traces using k-means clustering. The details of the Facebook and Yahoo! workloads that we use for evaluating our scheduler are provided in Table 4 [10]. We define heterogeneous users, with different minimum shares and priorities. Table 5 defines the users in Facebook and Yahoo! experiments. The minimum share of each user is defined based on its submitted job size, and each user submits jobs from one of the job classes in Table 4.

Users	*MinimumShare*	*Priority*
Facebook experiments		
U_1	5	1
U_2	0	2
U_3	0	2
U_4	5	1
U_5	10	2
U_6	15	1
U_7	4	2
U_8	10	1
U_9	10	1
U_{10}	15	1
Yahoo! experiments		
U_1	5	1
U_2	0	2
U_3	10	2
U_4	15	1
U_5	10	1
U_6	15	2
U_7	10	1
U_8	15	1

Table 5: User properties in experiments of both workloads

We submit 100 jobs to the system, which is sufficient to contain a variety of the behaviours in our Hadoop workload. 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 in our experiments.

5.3. Results

In each experiment we compare COSHH with the FIFO algorithm and the version of the Fair sharing algorithm presented in [1]. The comparison is based on the four performance metrics of interest, and the scheduling overheads.

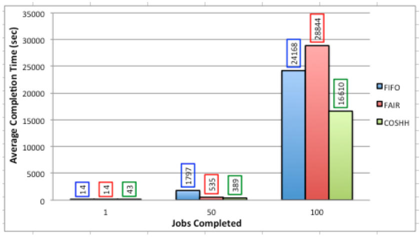

Figure 5: Average completion time for Facebook workload

Figures 5 and 6 present the average completion time metric for the algorithms running the Facebook and Yahoo! workloads, respectively. The results show that compared to the other algorithms, COSHH achieves the best

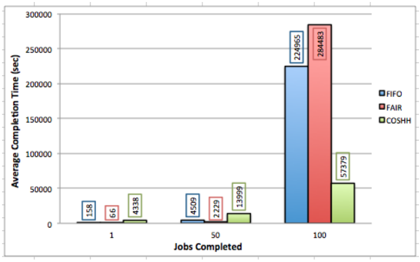

Figure 6: Average completion time for Yahoo! workload

Job Categories	Duration (sec)	Job	Input	Shuffle	Output	Map Time	Reduce Time
Facebook trace							
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
Yahoo! trace							
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 4: Job categories in both traces. Map time and Reduce time are in Task-seconds, e.g., 2 tasks of 10 seconds each is 20 Task-seconds.

average completion time for both workloads. This significant improvement can be explained by the fact that unlike the other two algorithms, COSHH considers the heterogeneity in making a scheduling decision based on the job requirements and the resource features. On the other hand, the Fair sharing algorithm has the highest average completion time compared to the other two algorithms. Both the Facebook and the Yahoo! workloads are heterogeneous, in which the arrival rates of small jobs are higher. In a heterogeneous Hadoop workload, jobs have different execution times (job sizes). 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. Therefore, when the Hadoop workload is heterogeneous, the FIFO algorithm can significantly increase the completion time of small jobs. 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 sizes of the jobs and the execution rates of the resources. In general, there is a considerable increase in completion times of all three algorithms after job 50. This is both because of the increased load in the system and also due to the presence of larger size jobs starting around this time.

As the Fair sharing algorithm does not have the problem of small jobs getting stuck behind large ones, we expect better average completion time for this scheduler than for the FIFO algorithm. However, because the Fair sharing algorithm first satisfies the minimum shares, it executes most of the small jobs after satisfying the minimum shares of the larger jobs. Therefore, the completion times of the small jobs (the majority of the jobs in this workload) are increased. On the other hand, the Fair sharing algorithm has the sticky slot problem, which arises when the scheduler assigns a job to the same resource at each heartbeat. This problem is first mentioned in [12] for the Fair sharing algorithm, where the authors considered the effect of this problem on locality. However, sticky slots can also significantly increase the average completion times, when an inefficient resource is selected.

In the Yahoo! workload, the COSHH algorithm leads to 74.49% and 79.73% improvement in average completion time over the FIFO algorithm, and the Fair sharing algorithm, respectively. Moreover, for the Facebook workload, the COSHH algorithm results in 31.27% and 42.41% improvement in average completion time over the FIFO algorithm, and the Fair sharing algorithm, respectively. It should be noted that the COSHH algorithm leads to a more substantial improvement for average completion time in the Yahoo! workload than in the Facebook workload. The reason is that the jobs in the Facebook workload are smaller and less heterogeneous than the jobs in the Yahoo! workload. As a result, taking the heterogeneity into account in the Yahoo! workload leads to greater improvement.

The overheads of the scheduling algorithms are presented in Figures 7 and 8 for the Yahoo! and the Facebook workloads, respectively. The improvement for average completion time in the COSHH scheduler is achieved at the cost of increasing the overhead of scheduling. However, the additional 5 second overhead for the COSHH algorithm, compared to the improvement for average completion time (which is more than 10000 seconds) is negligible. The time spent for classification and solving the LP at the beginning of the COSHH algorithm leads to a higher scheduling time. The scheduling times for both the Fair sharing and the COSHH algorithms increase considerably at around the point of scheduling the 50th to 60th jobs. These algorithms need to sort the users based on their shares to consider fairness and minimum share satisfaction. In the first stage of satisfying the minimum shares, they need to sort a smaller number of users. However, after satisfying the minimum shares, the number of users to be sorted is increased. Also, the order of users changes to consider fairness. As a result, the process of sorting users takes longer, and causes an increase in the scheduling time for both algorithms. The FIFO algorithm has the least overhead.

The significant feature of COSHH is that although it uses sophisticated approaches to solve the scheduling problem, it does not add considerable overhead. The reason is that first, we limit the number of times required to do classification, by considering aggregate measures of job features (i.e. mean execution time and arrival rate). Also, since some jobs in the Hadoop system are submitted multiple times by users, these jobs do not require changing the classification each time that they are submitted.

Fairness, dissatisfaction, and the locality of the algorithms are presented in Tables 6 and 7 for the Yahoo! and the Facebook workloads, respectively. The results for both workloads show that COSHH has competitive dissatisfaction and fairness with the Fair sharing algorithm. Because the COSHH scheduler has two stages to consider the minimum share satisfaction and fairness separately, it is successful in reducing the dissatisfaction along with improving the fairness. First the scheduler only satisfies the minimum shares based on the priority of the users, and then it focuses on improving fairness. Since COSHH considers the weights of the users, it does not let a high priority user with high minimum share starve lower priority users with smaller minimum shares.

Metrics	FIFO	Fair	COSHH
$Dissatisfaction$	8.618	$7.16E-04$	1.209
$Fairness$	4.974	0.965	2.779
$Locality(\%)$	95.6	97.4	96.5

Table 6: Dissatisfaction, fairness, and locality for Yahoo! workload

Metrics	FIFO	Fair	COSHH
$Dissatisfaction$	10.782	$8.31E-02$	0.294
$Fairness$	6.646	2.537	0.663
$Locality(\%)$	97.7	95.7	95.0

Table 7: Dissatisfaction, fairness, and locality for Facebook workload

The locality of COSHH is close to, and is in most cases better than the Fair sharing algorithm. This can be explained by the fact that our algorithm chooses the replication places based on the suggested classes for each resource. As our experimental environment is a small Hadoop cluster, our scheduler's replication method can not lead to considerable improvement in the locality. However, the advantages of using COSHH's replication method could be significant on large Hadoop clusters.

6. Analysis

In the previous section, we evaluated the algorithms in the case that the minimum share of each user is defined

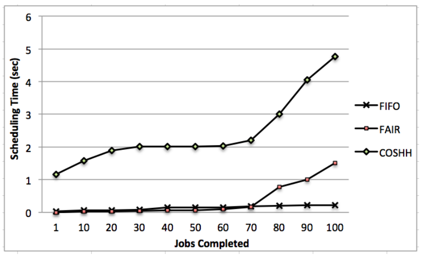

Figure 7: Scheduling overheads in Yahoo! workload

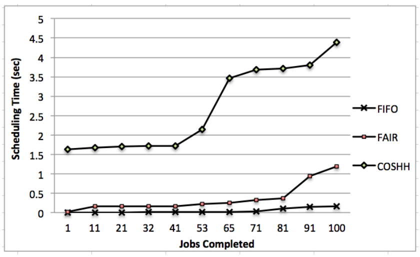

Figure 8: Scheduling overheads in Facebook workload

based on its submitted job size. However, as the minimum shares and the priorities of users are usually defined by the Hadoop provider, in some Hadoop systems the minimum shares may be defined without taking the job sizes into account. The performance of the Hadoop schedulers may be affected by different settings of minimum shares, scaling the number of jobs and resources, and any possible estimation errors. This section performs further evaluation on the COSHH scheduler and analyzes it from the points of view of minimum share effect, scalability and sensitivity.

6.1. Minimum Share Effect

Hadoop assigns a priority and a minimum share to each user, where the minimum share of a user defines the minimum number of slots that the system must provide for that user at each point in time. However, the priorities and minimum shares in each Hadoop system are defined based on a particular policy of the system such as the pricing policy in [6]. The minimum shares can help in improving average completion times, in particular when the minimum shares are defined based on the job sizes. So, in order to study the performance of the COSHH scheduler without the effect of minimum shares, here we study the performance of the algorithms with no minimum shares assigned to the users. The experimental environment in this part is the same as the previous section, except that the users are homogeneous with zero minimum shares, and priorities equal to one.

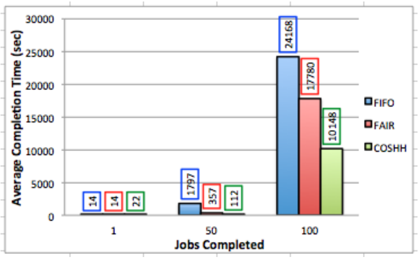

Figure 9: Average completion time for Facebook workload

Figures 9 and 10 present the average completion time metric for the algorithms running the Facebook and Yahoo! workloads, respectively. The results show that when the users are homogeneous, and no minimum share is defined, the average completion time of the FIFO algorithm is higher than the Fair sharing algorithm. Unlike the

FIFO algorithm, the Fair sharing algorithm does not have small jobs stuck behind large ones. In addition, the minimum share satisfaction of large jobs does not postpone the scheduling of smaller jobs. In the Yahoo! workload, the Fair sharing algorithm achieves a 37.72% smaller average completion time than the FIFO algorithm, and the COSHH algorithm reduces the average completion time of the Fair sharing algorithm by 75.92%. Moreover, in the Facebook workload, the Fair sharing algorithm achieves a 26.42% smaller average completion time than the FIFO algorithm, and the COSHH algorithm reduces the average completion time of the Fair sharing algorithm by 42.97%. Because of the job sizes, and level of heterogeneity, the average completion time improvement of COSHH for the Yahoo! workload is much higher than for the Facebook workload.

The overheads of the scheduling algorithms are presented in Figures 11 and 12 for the Yahoo! and the Facebook workloads, respectively. Because most of the jobs in this workload are small, and they have fewer tasks, the scheduling overheads are low. The classification and LP solution time in the COSHH algorithm lead to a large initial scheduling time. The overheads of both the Fair sharing and the COSHH algorithms are lower when the users are homogeneous. In this case these algorithms no longer have the minimum share satisfaction stages.

Fairness and locality of the algorithms are presented in Tables 8 and 9 for the Yahoo! and the Facebook workloads, respectively. Because there is no minimum share defined in this experiments, the amount of dissatisfaction for all schedulers is zero. The results show that the Fair sharing algorithm has the best, and the COSHH algorithm has the second best fairness.

Metrics	FIFO	Fair	COSHH
Fairness	1.032	0.504	0.856
Locality (%)	94.9	97.9	98.1

Table 8: Fairness and locality for Yahoo! workload

Metrics	FIFO	Fair	COSHH
Fairness	1.188	0.429	0.926
Locality(%)	93.4	95.2	98.3

Table 9: Fairness and locality for Facebook workload

The locality of COSHH is close to, and in most cases is better than the Fair sharing algorithm. This can be explained by the fact that our algorithm chooses the replication places based on the suggested classes for each resource.

6.2. Scalability Analysis

The Hadoop system can scale with respect to the number of resources in the Hadoop cluster, and the number of jobs submitted. It is critical for the Hadoop schedulers to scale accordingly. We evaluate the COSHH algorithm from scalability in two perspectives.

6.2.1. Number of jobs

In this part we evaluate the effect of the number of jobs in the Hadoop workload on performance of the COSHH algorithm. First, we consider a Hadoop workload with total job number of 5, and measure the performance of all schedulers in this underloaded system. Then, we increase the number of jobs in the Hadoop workload, and provide the results for each case. Figures 13 and 14 present the average completion times for the Yahoo! and the Facebook workloads, respectively.

When there are few jobs in the workload, and the system is very underloaded, the COSHH algorithm has the higher 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 in the queue until a slot becomes free. From this point on the improvement in average completion time for the COSHH algorithm overcomes its scheduling overhead.

When the system is highly underloaded, most of the resources are free. Therefore, a simple scheduling algorithm like FIFO which makes fast scheduling decisions leads to better average completion time. However, when the load in the system increases, the benefits of the COSHH algorithm result in better average completion time. In an overloaded system, initially the average completion time of the Fair sharing algorithm is better than the FIFO algorithm. After this initial transient, the average completion time of the FIFO algorithm becomes better. The reason is that the minimum shares assigned to the users are defined

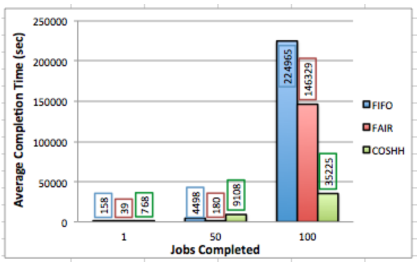

Figure 10: Average completion time for Yahoo! workload

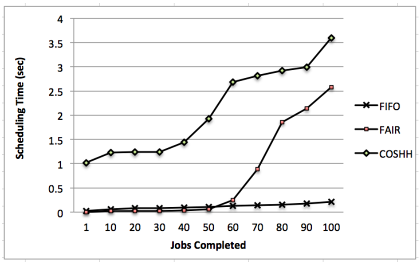

Figure 11: Scheduling overheads in Yahoo! workload

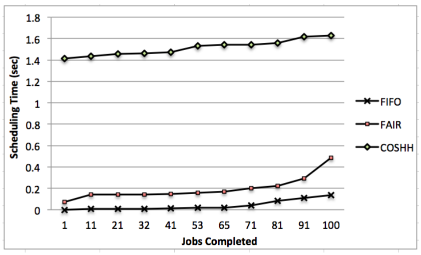

Figure 12: Scheduling overheads in Facebook workload

based on the job sizes. Initially, Fair sharing is satisfying the minimum shares, and as a result improves the average completion times. However, after minimum share satisfaction, the sticky slot feature leads to an increase in the average completion time.

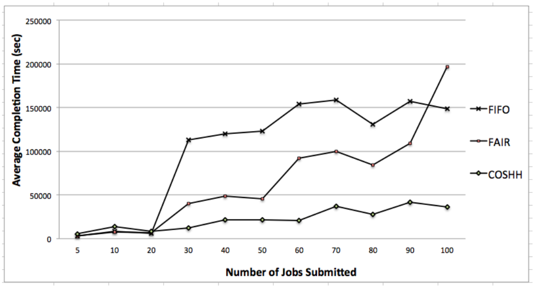

Figure 13: Average completion time for Yahoo! workload - Scalability based on number of jobs

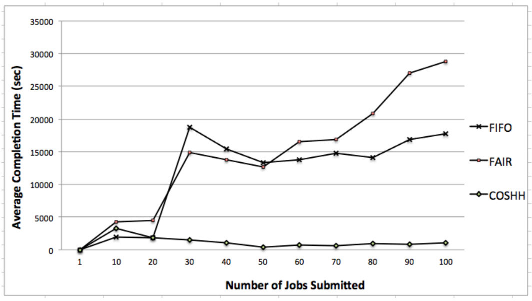

Figure 14: Average completion time for Facebook workload - Scalability based on number of jobs

The average completion time for the Fair sharing algorithm is initially low. However, once the load in the system increases and it is necessary to assign the submitted jobs to resources at different heartbeats, its average completion time increases at a greater rate than the others. The reason is that by increasing the number of jobs, at each heartbeat, the Fair sharing algorithm needs to perform sorting and searching over large sort and search spaces. Moreover, the sticky slot problem, and neglecting system heterogeneity lead to higher average completion time.

Figures 15 and 16 present the scheduling times for the Yahoo! and the Facebook workloads, respectively. The overheads of all of the algorithms increase as the number of

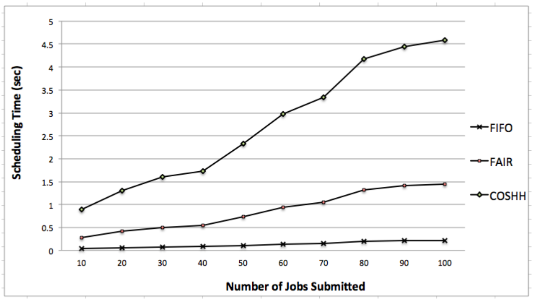

Figure 15: Scheduling time for Yahoo! workload - Scalability based on number of jobs

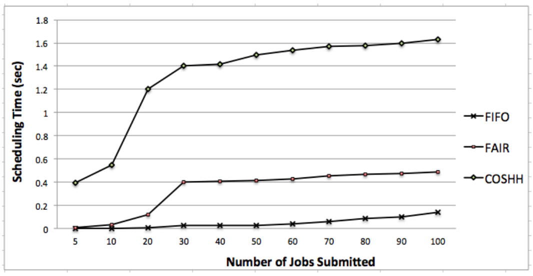

Figure 16: Scheduling time for Facebook workload - Scalability based on number of jobs

submitted jobs increases. The growth rate in 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. Typically, 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 has reduced overhead. In the Facebook workload, where the jobs sizes are smaller, increasing the number of jobs has less effect on the scheduling overhead.

6.2.2. Number of Resources

This part evaluates the performance of the schedulers when the number of resources varies. We first study a cluster with 6 resources, and then increase the number of resources. To define different size clusters, we use the six types of resources presented in Table 10. To increase the number of resources we add a new resource from one of the resource types in turn, starting from six resources, and increasing the number to 102 resources (i.e. 17 resources of each type).

Resources	Slot		Mem	
	slot#	*execRate*	*Capacity*	*RetrieveRate*
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 10: Resource Types

Figures 17 and 18 present the average completion times with various number of resources for the Yahoo! and the Facebook workloads, respectively. Increasing the number of resources reduces the load in the system and leads to reducing the average completion time for all schedulers. However, increasing the number of resources can reduce the chance of local execution, which leads to an increase in the average completion time. Therefore, by increasing the number of resources, first the average completion times of the schedulers reduce until the number of resources reaches approximately 57. Beyond this point the average completion times of the schedulers increase slightly, because of the locality issue.

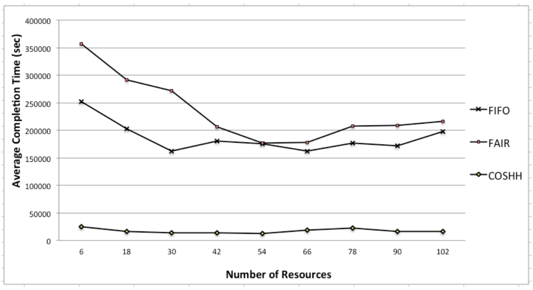

Figure 17: Average completion time for Yahoo! workload - Scalability based on number of resources

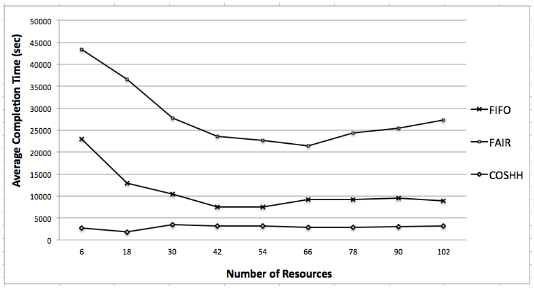

Figure 18: Average completion time for Facebook workload - Scalability based on number of resources

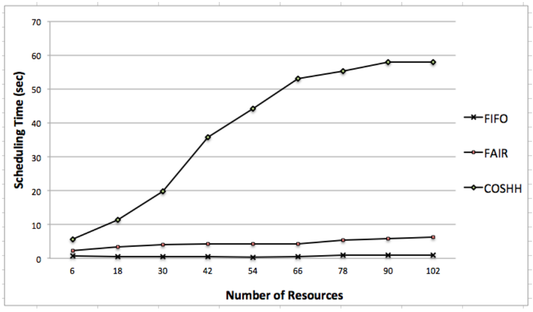

Figure 19: Scheduling time for Yahoo! workload - Scalability based on number of resources

Figures 19 and 20 present the scheduling times with various number of resources for the Yahoo! and the Facebook workloads, respectively. The overheads of the COSHH and the Fair sharing algorithms increase as the number of resources increases. The reason is that the search and sort times increase in these algorithms. Moreover, increasing the number of resources increases the classification and LP solution times in the COSHH algorithm. Therefore, the rate of increase in the COSHH algorithm is higher than the Fair sharing algorithm. However, its growth rate decreases as the number of resources increases. We remark that the total COSHH scheduling overhead here is around 60-180 seconds, which is negligible compared to the improvement for average completion time (which is around 20000-100000 seconds).

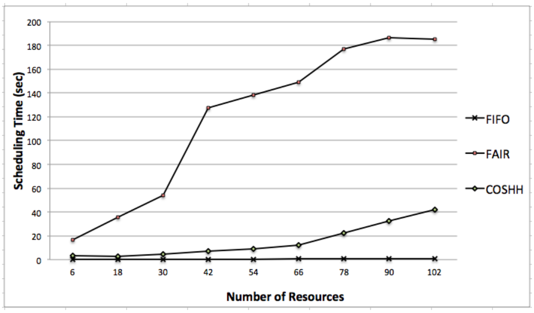

Figure 20: Scheduling time for Facebook workload - Scalability based on number of resources

6.3. Sensitivity Analysis

The Task Scheduler component in the COSHH scheduling system provides an estimate of the mean execution time of an incoming job. To measure how much the performance of our algorithm is dependent on the estimation error, we evaluate our algorithm with various workloads under different levels of error. In order to completely study the robustness of our algorithm, we examine cases that have 0% to 40% error in the estimates; typically these errors are on the order of 10% [15]. We use the error model discussed in [16] for estimating execution times. The error model in these estimates is an *Over and Under Estimation Error* model, which is as follows. Define the actual execution time of job i on Resource j to be $L(i,j)$. Let $\hat{L}(i,j)$ denote the (corresponding) estimated execution time. In our simulations, $\hat{L}(i,j)$ is obtained from $\hat{L}(i,j) = L(i,j) \times (1 + E_r)$. Here, E_r is the error for estimating the job execution time, which is sampled from the uniform distribution $[-I,+I]$, where I is the maximum error.

First, we evaluate our algorithm in an environment with accurate estimated execution times, and then increase the amount of error. Figures 21 and 22 present the average completion times for the Yahoo! and the Facebook workloads, respectively. Different error levels are considered in these figures, where here COSHH-n denotes the COSHH scheduler with n% error in estimating the execution times. For each experiment, we run 30 replications in order to construct 95 percent confidence intervals. The lower and upper bounds of the confidence intervals are represented with lines on each bar.

Based on the results, up to 40 percent error in estimation does not significantly affect the average completion time of our algorithm. The reason is that the COSHH algorithm uses the estimated execution times to provide suggestions for each resource; the estimates themselves are not directly used in the final scheduling decisions. The small levels of error lead to only a slight change in job classes. As a result, the average completion time is not increased for 10% error levels. When the error level is higher, it leads to more changes in the job classes, which modifies the suggested set of classes considerably, and affects the average completion time. The results show that the COSHH algorithm is not too sensitive to the estimated execution times, and it maintains better average completion time than the other algorithms in up to 40% error in estimating the execution times.

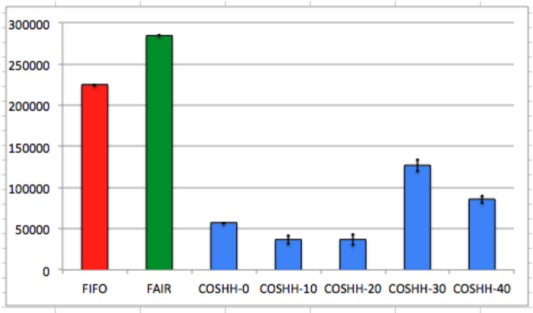

Figure 21: Average completion time for Yahoo! workload - Sensitivity to error in estimation

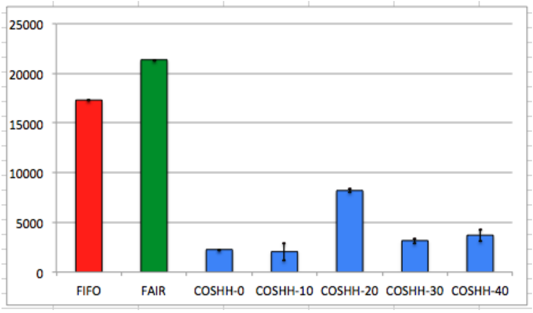

Figure 22: Average completion time for Facebook workload - Sensitivity to error in estimation

Figures 23 and 24 give the scheduling times for the Yahoo! and the Facebook workloads, respectively. The error level can cause a slight increase in scheduling time, which is caused by longer classification processes. When there are estimation errors, the k-means clustering method may need to run more steps to reach the appropriate classification.

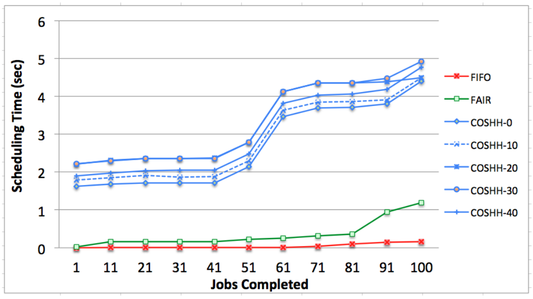

Figure 23: Scheduling time for Yahoo! workload - Sensitivity to estimation error

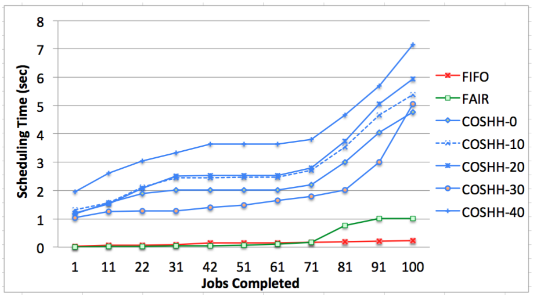

Figure 24: Scheduling time for Facebook workload - Sensitivity to estimation error

7. Results on Real Hadoop System

In this section the performance of Hadoop schedulers is evaluated by running experiments using a real Hadoop workload on a Hadoop cluster. Due to our limitations in evaluation on a real cluster, the results should be seen as a basis for verifying overheads and practicality of the solution on real systems. The simulation results provide a wider possibility of evaluation at differing levels of heterogeneity, scale and sensitivity. The jobs in these experiments are selected from the Facebook workload used in previous sections, and are presented in Table 4. The users are defined to be homogeneous with zero minimum shares, and priorities equal to one. A cluster of four quad core nodes is used for these experiments (Table 11). The bandwidth between the resources is 2Gbps. Hadoop 0.20 is installed on the cluster, and the Hadoop block size is set to 128MB. Also, the data replication number is set to the default value of three in all algorithms.

Resources	Slot		Mem	
	$slot\#$	$execRate$	$Capacity$	$RetrieveRate$
R_1	4	$100MHz$	$500MB$	$3.2GB/s$
R_2	4	$800MHz$	$16GB$	$3.2GB/s$
R_3	4	$400MHz$	$4GB$	$3.2GB/s$
R_4	4	$3200MHz$	$32GB$	$3.2GB/s$

Table 11: Resources in the heterogeneous cluster

Table 12 presents the dissatisfaction performance metric for the schedulers. As the Fair Sharing algorithm has minimum share satisfaction as its main goal, it is successful in reducing the dissatisfaction rate compared to the other schedulers. The COSHH algorithm also considers the minimum shares as the first critical issue in making scheduling decisions. It assigns the minimum shares while it takes the system heterogeneity into account. This results in competitive dissatisfaction with the Fair Sharing algorithm. However, the FIFO algorithm significantly increases the dissatisfaction rate by ignoring the minimum shares.

FIFO	Fair	COSHH
10.302	3.984	5.869

Table 12: Dissatisfaction of schedulers - Heterogeneous Users.

In Figure 25, the average completion times for the three schedulers are presented. The FIFO algorithm achieves better average completion time than the Fair Sharing algorithm, when the users are set to be heterogeneous. Both the FIFO and the Fair Sharing algorithms do not consider heterogeneity in their scheduling decisions. However, the minimum shares effect on the Fair Sharing algorithm leads to larger average completion time. When the users are homogeneous the average completion time of the FIFO algorithm becomes larger than the Fair Sharing algorithm. In this case, the starvation of small jobs behind large jobs in the FIFO scheduler increases the average completion time of small jobs, and leads to larger average completion time.

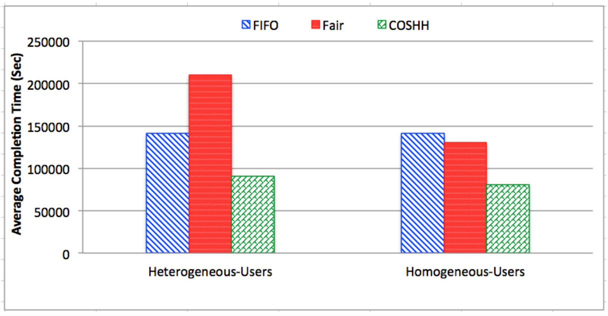

Figure 25: Average completion time of schedulers.

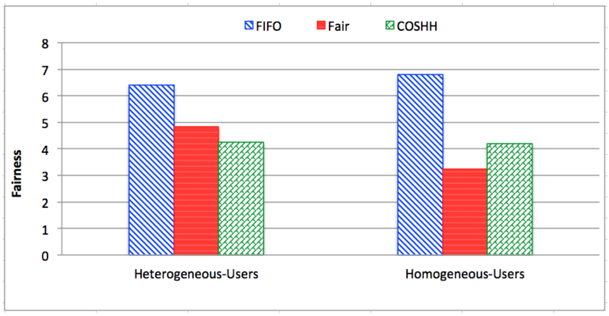

Figure 26: Fairness of schedulers.

Figures 26 and 27 present the fairness and locality of the schedulers, respectively. The Fair Sharing algorithm provides the best fairness, while the COSHH scheduler has competitive locality and fairness. The scheduling overheads in the case studies are presented in Figure 28. Scheduling complexity for the COSHH algorithm leads to higher scheduling time and overhead. However, the total scheduling overhead of COSHH is less than 25 seconds, which is negligible compared to the processing times.

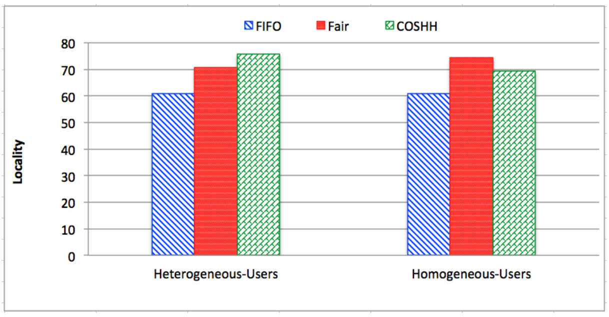

Figure 27: Locality of schedulers.

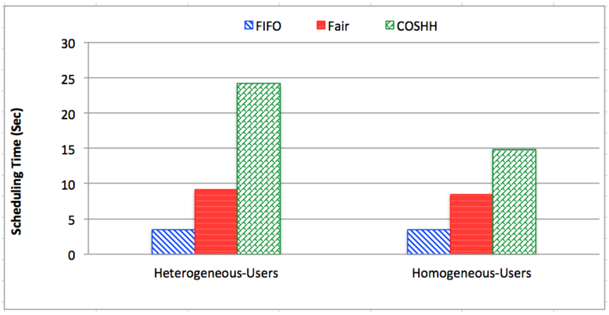

Figure 28: Scheduling time of schedulers.

8. Related Work

In this section we present an overview of some of the current Hadoop scheduling algorithms. More detailed comparison of Hadoop schedulers with schedulers introduced for other distributed computing systems are provided in the first author's PhD thesis [9].

MapReduce was initially designed for small clusters in which a simple scheduling algorithm like FIFO can achieve an acceptable performance level. However, experience from deploying Hadoop in large systems shows that simple scheduling algorithms like FIFO can cause severe performance degradation; particularly in systems that share data among multiple users [1]. As a result, the next generation scheduler in Hadoop, Hadoop on Demand (HOD) [17], addresses this issue by setting up private Hadoop clusters on demand. 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. To address some of the shortcomings of the FIFO algorithm, Hadoop added a scheduling plug-in framework with two additional schedulers that extend rather than replace the original FIFO scheduler [1].

The additional schedulers are introduced in [1], where they are collectively known as Fair sharing. Fair sharing defines a pool for each user, each pool consisting of a number of map slots 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 the other users to speed up the other jobs in the system. The Fair sharing algorithm does not achieve good performance regarding locality [12]. Therefore, in order to improve the data locality, a complementary algorithm for Fair sharing is introduced in [12], called delay scheduling. Using the delay scheduling algorithm, when Fair sharing chooses a job for the current free resource, and the resource does not contain the required data for the job, scheduling of the chosen job is postponed, and the algorithm finds another job. However, to limit the waiting time, a threshold is defined; if scheduling of a job remains postponed until the threshold is met, the job will be submitted to the next free resource. The proposed algorithms can perform much better than Hadoop's default scheduling algorithm (FIFO); however, these algorithms do not consider heterogeneous systems in which resources have different capacities and users submit various types of jobs.

In [6], a Dynamic Priority (DP) parallel task scheduler is designed for Hadoop, which allows users to control their allocated capacity by dynamically adjusting their budgets. This algorithm prioritizes users based on their spending, and allows capacity distribution across concurrent users to change dynamically based on user preferences. The core of this algorithm is a proportional share resource allocation mechanism that allows users to purchase or to be granted a queue priority budget. This budget may be used to set spending rates denoting the willingness to pay a certain amount per Hadoop map or reduce task slot per time unit.

Dominant Resource Fairness (DRF) [2] addresses the problem of fair allocation of multiple types of resources to users with heterogeneous demands. In particular, the proposed algorithm is a generalization of max-min fairness for multiple resources. The intuition behind DRF is that in a multi-resource environment, the allocation of a user should be determined by the user's dominant share, which is the maximum share of any resource that the user has been allocated. In a nutshell, DRF seeks to maximize the minimum dominant share across all users. For example, if user A runs CPU-heavy tasks and user B runs memory-heavy tasks, DRF attempts to equalize user A's share of CPUs with user B's share of memory. In the single resource case, DRF reduces to max-min fairness for that resource. This algorithm only considers heterogeneity in the user's demand level. It does not take into account resource heterogeneity in making scheduling decisions.

9. Future Work

Heterogeneity is for the most part neglected in designing Hadoop scheduling systems. In order to keep the algorithm simple, minimal system information is used in making scheduling decisions, which in some cases could result in poor performance. Growing interest in applying the MapReduce programming model in various applications gives rise to greater heterogeneity, and thus must be considered in its impact on performance. It has been shown that it is possible to estimate system parameters in a Hadoop system. Using the system information, we designed a scheduling algorithm which classifies the jobs based on their requirements and finds an appropriate matching of resources and jobs in the system. Our algorithm is completely adaptable to any variation in the system parameters. The classification part detects changes and adapts the classes based on the new system parameters. Also, the mean job execution times are estimated when a new job is submitted to the system, which makes the scheduler adaptable to changes in job execution times.

For future work, we plan to improve the performance of our scheduler by separating data intensive and computation intensive jobs in performing the classification.

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 Ion Stoica and Sameer Agarwal for sharing the task scheduler part, and also their comments on our proposed scheduler.

References

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

[2] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, 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. 24–37.
URL `http://dl.acm.org/citation.cfm?id=1972457.1972490`

[3] K. Morton, M. Balazinska, D. Grossman, ParaTimer: a progress indicator for MapReduce DAGs, in: Proceeding of the International Conference on Management of Data, 2010, pp. 507–518.

[4] S. Agarwal, I. Stoica, Chronos: A predictive task scheduler for MapReduce, Tech. rep., EECS Department, University of California, Berkeley, Author1 URL: `http://www.cs.berkeley.edu/∼sameerag/`, Author1 email: `sameerag@cs.berkeley.edu` (December 2010).

[5] S. Hammoud, M. Li, Y. Liu, N. K. Alham, Z. Liu, MRSim: a discrete event based MapReduce simulator, in: Proceedings of the 7th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2010), Yantai, Shandong, China, 2010, pp. 2993–2997.

[6] T. Sandholm, 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.

[7] C. Mair, G. F. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. J. Shepperd, S. Webster, An Investigation of Machine Learning based Prediction Systems, Journal of Systems and Software 53 (1) (2000) 23–29.

[8] A. Ethem, Introduction to Machine Learning (Adaptive Computation and Machine Learning), The MIT Press, 2004.

[9] A. Rasooli, Improving scheduling in heterogeneous Grid and Hadoop systems, Ph.D. thesis, McMaster University, Hamilton, Canada (July 2013).

[10] Y. Chen, A. Ganapathi, R. Griffith, 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, Washington, DC, USA, 2011, pp. 390–399.

[11] R. Bodkin, Yahoo! updates from Hadoop Summit 2010, `http://www.infoq.com/news/2010/07/yahoo-hadoop-summit` (July 2010).

[12] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, 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.

[13] IBM ILOG CPLEX optimizer, [Online; accessed 30-November-2010] (2010).
URL `http://www-01.ibm.com/software/integration/optimization/cplex/`

[14] IBM ILOG CPLEX development bundle, [Online; accessed November-2010] (2009).
URL `http://www-01.ibm. com/software/integration/optimization/cplex-dev-bundles/`

[15] S. Akioka, Y. Muraoka, Extended forecast of cpu and network load on computational grid, in: Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid(CCGrid'04), IEEE Computer Society, Los Alamitos, CA, USA, 2004, pp. 765–772.

[16] A. Iosup, O. Sonmez, S. Anoep, D. Epema, The performance of bags-oftasks in large-scale distributed systems, in: Proceedings of the 17th International Symposium on High Performance Distributed Computing, 2008, pp. 97–108.

[17] Apache, Hadoop on demand documentation, [Online; accessed 30-November-2010] (2007).
URL `http://hadoop.apache.org/common/docs/r0.17.2/hod.html`