Heuristics for Periodical Batch Job Scheduling in a MapReduce Computing Framework

Xiaoping Li[a,b,*], Tianze Jiang[a,b], Rubén Ruiz[c]

[a] *School of Computer Science and Engineering, Southeast University, Nanjing 211189, China*

[b] *Key Laboratory of Computer Network and Information Integration, Ministry of Education, Nanjing, 211189, China*

[c] *Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.*

Abstract

Task scheduling has a significant impact on the performance of the MapReduce computing framework. In this paper, a scheduling problem of periodical batch jobs with makespan minimization is considered. The problem is modeled as a general two-stage hybrid flow shop scheduling problem with schedule-dependent setup times. The new model incorporates the data locality of tasks and is formulated as an integer program. Three heuristics are developed to solve the problem and an improvement policy based on data locality is presented to enhance the methods. A lower bound of the makespan is derived. 150 instances are randomly generated from data distributions drawn from a real cluster. The parameters involved in the methods are set according to different cluster setups. The proposed heuristics are compared over different numbers of jobs and cluster setups. Computational results show that the performance of the methods is highly dependent on both the number of jobs and the cluster setups. The proposed improvement policy is effective and the impact of the input data distribution on the policy is analyzed and tested.

Keywords: MapReduce, Periodical job, Schedule-dependent setup times, Heuristics, Makespan

*Corresponding author: Dr. Xiaoping Li, Professor with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. Tel.& Fax: 86-25-52090916.

Email addresses: `xpli@seu.edu.cn` (Xiaoping Li), `rruiz@eio.upv.es` (Rubén Ruiz)

Preprint submitted to Elsevier *June 16, 2015*

1. Introduction

Huge attention has been paid on Big Data from researchers in information sciences, policy and decision makers in governments and enterprises [24, 27, 8]. MapReduce [9] is a programming and implementation framework model for processing large data sets (in the order of petabytes in size) with parallel and distributed algorithms that run on clusters. It has emerged as a leading distributed computing framework for large-scale data processing including: web crawling, data mining, recommendation systems and log analysis among others. Apache Hadoop [40] is a popular open-source implementation of the MapReduce framework provided by the Apache Software Foundation. The MapReduce model consists of the Map() procedure, which carries out a selection, filtering or sorting of the data and the Reduce() method which processes and summarizes the information. The whole framework is in charge of the processing by providing marshalling of the distributed computers, parallelizing the tasks, managing communications between nodes and dealing with redundancy and tolerance to faults. As such, MapReduce implementations are the backbone of many existing Big Data and Cloud efforts by large companies.

Designers and users pay close attention to the performance of MapReduce since they ordinarily have diverse performance metrics and requirements such as job response time, throughput, and sharing of cluster and resource utilization that are highly dependent on task scheduling. However, different scenarios need appropriate task scheduling policies so that various performance metrics are optimized. In general, MapReduce task scheduling can be *on-line* and *off-line*.

On-line task scheduling mainly focuses on job performance and resource utilization. As regards proposals of models for job performance measures and dynamic scheduling, Polo et al. [26] proposed an estimator to predict job completion times according to the job progress. The scheduler relies on estimates of individual job completion times given a particular resource allocation, and uses these estimates to maximize each job's chances of meeting its performance goal. Verma et al. [37] offered a new resource sizing and provisioning service in MapReduce providing a set of provisioning options according to past job executions and the user's performance goal. The Fair Scheduler [42] considers two problems concerning MapReduce jobs: data locality and Map/Reduce interdependency. Delay scheduling as well as the copy-compute splitting policy is developed to address the problems. Later, Zaharia et al. [43] observed the conflict between fairness and data locality, for which a simple algorithm called delay was proposed. FLEX [41] is an extension of the Fair Scheduler which considers a variety of metrics such as the response time and makespan. Berlińska and Drozdowski [3] analyzed MapReduce distributed computations as a divisible load scheduling problem. A divisible load model of the computation and two load partitioning algorithms were proposed. Regarding the adjustment of resource allocation considering job profiling and node performance checking to optimize resource utilization, Tian et al. [35] classified the MapReduce workloads into three categories based on their CPU and I/O usage, with which a three-queue scheduler was proposed to improve both CPU and I/O utilization. Asahara et al. [1] presented a locality and an I/O load-aware task scheduler to mitigate the I/O bottlenecks of a cluster with locality and an I/O load-aware map task assignment and storage selection. Lu et al. [19] designed the Workload Characteristic Oriented Scheduler which strives to co-locating tasks of possibly different MapReduce jobs with complementary resource usage characteristics. Shih et al. [33] proposed a dynamic slot-based task scheduling scheme by considering the physical workload on each node so as to prevent resource underutilization. Zaharia et al. [44] showed that traditional task schedulers cause performance degradation in heterogeneous environments and proposed LATE scheduling algorithms, robust to heterogeneity and to improve response times. Chen et al. [6] proposed a self-adaptive MapReduce scheduling algorithm which calculates the progress of tasks dynamically, and automatically adapts to the continuously varying environment.

As for off-line scheduling, state-of-the-art works consider modeling MapReduce task scheduling as a classic scheduling problem. Chang et al. [5] first presented a simplified abstraction of the MapReduce scheduling problem, and then formulated the problem as a linear program. Various on-line and off-line algorithms were developed to minimize the overall job completion times. Phan et al. [23] formulated the off-line scheduling of real time MapReduce jobs on a heterogeneous Hadoop architecture as a constraint satisfaction problem and introduced various search strategies for it. Fischer et al. [10] proposed an idealized Hadoop model to investigate the Hadoop task assignment problem. A round-robin method and a flow-based algorithm were presented to compute the assignments. Moseley et al. [21] formulated job scheduling in MapReduce as a generalization of the two-stage classical flexible flow shop problem minimizing total flowtime. In addition various approximation algorithms were investigated for both off-line and on-line scheduling.

In MapReduce production clusters, some independent batch jobs are periodically executed [34] on new data, of which the properties can be obtained by analyzing a job's historical information. With these properties, the schedule of a set of jobs is generated optimizing a given performance goal, for example, minimizing the makespan. With the knowledge of the execution period, the release times of jobs are determined and therefore this scenario is also off-line. Verma et al. [38] considered the above problem as a classical two-stage flow shop problem minimizing the makespan. They began with Johnson's algorithm [14] to solve the problem. Then a balanced pool heuristic method was proposed considering the defects of the classical model. The heuristic relies on a MapReduce simulator [36]. Recently, Wang & Shi [39] proposed task-level scheduling algorithms with respect to budget and deadline constraints for a batch of MapReduce jobs on a set of provisioned heterogeneous machines in cloud platforms. The batch of jobs were organized as a k-stage workflow and two related optimization problems were considered.

In this paper, we consider the scheduling problem of periodical batch jobs in MapReduce which is rarely studied with the exception of [38]. Data locality is an important factor that affects task scheduling but is seldom considered in the model. We measure data locality by the time that tasks spend on inputting data. Since a task's setup time depends not only on the data size and data locality but also on the schedule of other tasks, it could be regarded as a schedule-dependent setup time. Furthermore, the feature of parallel multi-tasks in each phase is fully taken into account as well as the pipelined fashion of the map and reduce phase. We model the problem as a general hybrid flow shop, which is more practical than that considered in [21]. The problem is thus converted to a general two-stage hybrid flow shop scheduling problem with schedule-dependent setup times and is formulated using integer programming. To the best of our knowledge the problem has never been considered with these extensions, which results in a much more practical and close to real life model. The objective is to minimize the makespan and some heuristics are proposed for the considered problem. We present some tight lower bounds that are used to test the effectiveness of the presented heuristics.

The rest of the paper is organized as follows. Section 2 contains a detailed description of the problem considered and formulates it as an integer program. A lower bound of the makespan is described in Section 3. Section 4 describes the proposed heuristic methods. Experimental results are presented in Section 5. Section 6 concludes the paper and gives further research directions.

2. Problem description

The notations employed in the following are detailed in Table 1.

Table 1: Notation employed in the paper.

Q	a MapReduce cluster
Q_m	the set of all map slots in Q with size M_m
Q_r	the set of all reduce slots in Q with size M_r
$\mathbb{J}$	the set of MapReduce jobs $\mathbb{J} = \{J_1, J_2, \ldots, J_n\}$
a	$a \in \{m, r\}$ denotes either the map phase or the reduce phase
V_i^a	the task set of job J_i in phase a
$v_{i,j}^a$	the task j in V_i^a
T_a	the set of all tasks of the jobs in $\mathbb{J}$ in phase a, $T_a = \bigcup\limits_{i=1}^{n} V_i^a$
$p_{i,j}^a$	the processing time of task $v_{i,j}^a$ executed by slot of Q_a
$s_{i,j}^a$	the setup time before $p_{i,j}^a$ for input data
$s_{i,j,k}^a$	the setup time of task $v_{i,j}^a$ processed on slot k
$b_{i,j}^a$	the start time of task $v_{i,j}^a$
$c_{i,j}^a$	the completion time of task $v_{i,j}^a$

Usually, there are five phases in MapReduce: Preparation (input involved data), Map (filtering and sorting the data), Shuffle (redistribute the mapped data), Reduce (process each group of the redistributed data), and Output (collect all the Reduce output). Because input data is usually large, in the order of petabytes, it is processed on MapReduce clusters. Generally, a MapReduce cluster Q contains many nodes. There is one or more slot(s) in each node (a physical or virtual machine). Q_m is the set of all map slots in Q with size M_m; Q_r is the set of all reduce slots with size M_r. Each slot type can be regarded as a group of identical machines. For a set of n MapReduce jobs $\mathbb{J} = \{J_1, J_2, \ldots, J_n\}$, each job in $\mathbb{J}$ is submitted to Q for processing successively in map and reduce phases. $a \in \{m, r\}$ denotes a phase. m represents the map phase and r the reduce phase. The task set of job J_i in phase a is V_i^a, in which task j is denoted as $v_{i,j}^a$. Let T_a be the set of all tasks of the jobs in $\mathbb{J}$ in phase a, i.e., $T_a = \bigcup\limits_{i=1}^{n} V_i^a$. The following assumptions and constraints are considered for clustering and task execution:

(i) A MapReduce cluster is homogeneous and the number of slots in each node is configured as the CPU core number. There is no node or task failure during execution.

(ii) Task processing times are known in advance and are obtained from historical executions. The distribution and size of input data for the map task is also of prior knowledge. For each reduce task, the size of data read from each map task is equal.

(iii) There is no overlapping between the map and reduce phase for each job, implying that the reduce phase cannot start until the map phase has completed. The release times of all map tasks are set to 0 and the release time of a reduce task is the latest completion time of all map tasks from the same job.

(iv) No slot can process more than one task at any time; no task can be processed by more than one slot at the same time. Each slot starts processing the next task without waiting once the current task is finished.

Task $v^a_{i,j}$ can be executed by any slot of Q_a with the processing time $p^a_{i,j}$, requiring the setup time $s^a_{i,j}$ for input data. Generally, $s^a_{i,j}$ is affected by three factors: the data size, data locations and communication rates among nodes. Setup times are schedule-dependent [20] since they depend on slot selection in each phase, i.e., they vary with the processing slots. Let $s^a_{i,j,k}$ be the setup time of task $v^a_{i,j}$ processed on slot k. For a given slot k, parameters of the three factors are determined which imply that $s^a_{i,j} = s^a_{i,j,k}$. Let $b^a_{i,j}$ and $c^a_{i,j}$ be the start and completion time of task $v^a_{i,j}$. It follows that $c^a_{i,j} = b^a_{i,j} + s^a_{i,j} + p^a_{i,j}$. Each slot obtains a sequence with tasks to process after a schedule is generated. A feasible schedule π is determined by the start time of each task while meeting the constraints and assumptions above. For $\mathbb{J}$ and cluster Q, the target of the considered problem is to generate a feasible schedule π minimizing makespan $C_{\max} = \max\limits_{\substack{i\in\{1,2,\ldots,n\} \\ j\in\{1,2,\ldots,|V^r_i|\}}} c^r_{i,j}$.

The two-stage hybrid flow shop problem (HFSP) is a typical scheduling problem: a number of jobs are processed on two stages, each job is processed first on stage I and then stage II. There are more than one identical machines in every stage. Jobs have to be assigned to exactly one machine at stage. The sequences of jobs at each machine at both stages have to be optimized. The problem considered is more general than a traditional HFSP because each job is divided into several tasks in each phase which indicates that each job is processed by a number of slots rather than only one slot (machine) in a HFSP. If each job has a single task in each phase, the problem considered resembles a hybrid flow shop. In any case, we are also considering the schedule-dependent setup times so the scheduling setting considered in this paper is original as far as the scheduling literature is concerned and to the best of our knowledge. A careful examination of the two recent reviews on the state-of-the-art of the hybrid flow shop literature by [30] and [32] and the references therein confirm this conclusion. Figure 1 shows an example Gantt chart for MapReduce task scheduling. The shadowed parts denote setup times.

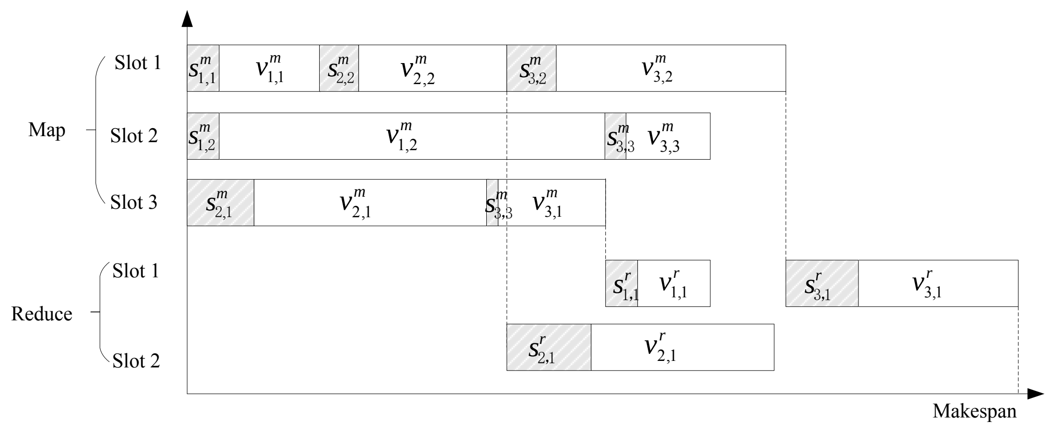

Figure 1: Gantt chart for MapReduce task scheduling.

We construct an integer program for the considered problem. To make sure that each task in a slot's task sequence has a predecessor and a successor, we place two dummy tasks v_h and v_t before the first task and after the last task in each slot, respectively. The setup times and processing times of these two dummy tasks are 0. The decision variables needed are defined as:

$$u^k_{v,v'} = \begin{cases} 1 & \text{if task } v \text{ is the immediate predecessor of task } v' \\ & \text{in slot } k\text{'s} \\ 0 & \text{otherwise} \end{cases}$$

The problem is formulated as follows:

$$\min \quad C_{\max} = \max_{\substack{i \in \{1,2,\ldots,n\} \\ j \in \{1,2,\ldots,|V^r_i|\}}} c^r_{i,j} \tag{1}$$

s.t.

$$c^m_{i,j} \geq s^m_{i,j} + p^m_{i,j}, \ \forall i \in \{1,2,\ldots,n\}, \ j \in \{1,2,\ldots,|V^m_i|\} \tag{2}$$

$$c^r_{i,j} \geq s^r_{i,j} + p^r_{i,j} + c^m_{i,l},\ \forall i \in \{1,2,\dots,n\},\ l \in \{1,2,\dots,\\ |V^m_i|\},\ j \in \{1,2,\dots,|V^r_i|\} \tag{3}$$

$$s^a_{i,j} = \textstyle\sum_{k \in Q_a} \sum_{v \in T_a \bigcup \{v_t\}} s^a_{i,j,k} u^k_{v^a_{i,j},v},\ \forall a \in \{m,r\},\\ i \in \{1,2,\dots,n\},\ j \in \{1,2,\dots,|V^a_i|\} \tag{4}$$

$$c^a_{i,j} - c^a_{i',j'} \geq s^a_{i,j} + p^a_{i,j} + \mathcal{M}(\textstyle\sum_{k \in Q_a} u^k_{v^a_{i',j'},v^a_{i,j}} - 1),\\ i,i' \in \{1,2,\dots,n\},\ j \in \{1,2,\dots,|V^a_i|\},\ j' \in \{1,2,\dots,|V^a_{i'}|\} \tag{5}$$

$$\textstyle\sum_{k \in Q_a} \sum_{v \in T_a \bigcup \{v_h\}} u^k_{v,v'} = 1,\ \forall v' \in T_a \tag{6}$$

$$\textstyle\sum_{k \in Q_a} \sum_{v' \in T_a \bigcup \{v_t\}} u^k_{v,v'} = 1,\ \forall v \in T_a \tag{7}$$

$$\textstyle\sum_{v' \in T_a \bigcup \{v_t\}} u^k_{v_h,v'} = 1,\ \forall k \in Q_a \tag{8}$$

$$\textstyle\sum_{v \in T_a \bigcup \{v_h\}} u^k_{v,v_t} = 1,\ \forall k \in Q_a \tag{9}$$

$$\textstyle\sum_{v \in T_a \bigcup \{v_h\}} u^k_{v,v'} = \sum_{v \in T_a \bigcup \{v_t\}} u^k_{v',v},\ \forall k \in Q_a,\ v' \in T_a \tag{10}$$

$$u^k_{v,v} = 0,\ \forall v \in T_a,\ k \in Q_a \tag{11}$$

$$u^k_{v,v'} \in \{0,1\},\ \forall v,v' \in T_a \textstyle\bigcup \{v_h,v_t\},\ k \in Q_a \tag{12}$$

Equations (2)-(3) provide extra constraints for task completion times. Constraint (2) ensures that the completion time of any map task is no less than the sum of its setup time and processing time. Constraint (3) assures that the completion time of any reduce task is no less than the sum of its setup time, processing time and the maximum completion time of map tasks from the same job as the reduce phase cannot start before the map phase completes. Constraint (4) calculates the real setup time for each task. Constraint (5) states that for phase a, if task $v^a_{i,j}$ and $v^a_{i',j'}$ are scheduled in the same slot and $v^a_{i',j'}$ is the immediate predecessor of $v^a_{i,j}$ in the slot's task sequence, $v^a_{i,j}$ cannot start processing until $v^a_{i',j'}$ is finished, which implies that each slot is prohibited from processing more than one task simultaneously. $\mathcal{M}$ is set to a very large constant, greater than the sum of all job processing times and setup times. Constraints (6)-(7) ensure that there is one and only one task scheduled in each position in a slot's task sequence. Each task has one and only one immediate predecessor and one immediate successor. Constraints (8)-(9) ensure that only one task is assigned to the first and last positions in each slot. Constraint (10) states that if a task has an immediate predecessor task in a sequence, it must have an immediate successor task and vice-versa. Constraint (11) assures that a task cannot be its own predecessor or successor. Constraint (12) specifies the nature of the decision variables. Note that according to the papers reviewed in [30] and [32] regarding mathematical models proposed for related hybrid flow shop problems, there is very little hope of solving the previous model to optimality even for small instance sizes. Since typical workloads of MapReduce clusters involve hundreds of tasks, such a model would result in tens of thousands of binary variables, motivating the need for heuristic methods.

3. Lower bounds

The two-stage hybrid flow shop problem (HFSP) is NP-hard even if the number of machines at one of the two stages is one [11]. The problem considered is also NP-hard because of the complexity over HFSP in that each job contains multiple tasks at each phase and each task has a schedule-dependent setup time. It is fairly difficult to find an optimal solution in an acceptable time for large problems. Instead, we present a tight lower bound that, similar to [12, 18, 29], is used to evaluate the relative performance of the proposed heuristic methods. The lower bound of the makespan for a two-stage hybrid flow shop problem is loosely based on that of Haouari and M'Hallah [12]. We propose two lower bounds as follows.

Let $\min_{[k]}$ denote the k^{th} minimal value (so $\min_{[1]}$ is the minimum value) in a non-increasing sequence.

Definition 1 *For task $v^a_{i,j}$, the artificial processing time $L^a_{i,j}$ is the sum of processing time and the minimum possible setup time, i.e., $L^a_{i,j} = p^a_{i,j} + \min_{k\in Q_a}\{s^a_{i,j,k}\}$. Function $h_a(x)$ returns the sum of the last x artificial tasks with processing times $L^a_{i,j}$ at phase a, i.e., $h_a(x) = \sum\limits_{k=1}^{x} \min_{[k]} L^a_{i,j}$ $(i \in \{1, 2, \ldots, n\},\ j \in V^a_i,\ a \in \{m, r\})$.*

Theorem 1 $LB_1 = \max\left\{\frac{h_m(M_r)}{M_r} + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^r|} L^r_{i,j}}{M_r},\right.$ $\left.\frac{h_r(M_m)}{M_m} + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^m|} L^m_{i,j}}{M_m}\right\}$ *is a lower bound of makespan of any feasible solution.*

Proof An intuitive lower bound LB' of $C_{\max}$ is the average of the total available time I_r and the total processing time P_r on all reduce slots at the reduce phase, i.e.,

$$LB' = \frac{1}{M_r}(I_r + P_r) \leq C_{\max} \tag{13}$$

An earlier finish at the map phase means an earlier start at the reduce phase. A lower bound of the total available time of the reduce slots is the total completion time of the only M_r map tasks with the minimal $L^m_{i,j}$ values, of which the minimum can be obtained by the Shortest Processing Time first (SPT) rule [4]. In fact, task $v^a_{i,j}$ spends $L^a_{i,j}$ on the slot allocated to it. So when setup times take the minimum, the sum of $L^r_{i,j}$ of all reduce tasks is a lower bound of total processing time of reduce slots. We obtain:

$$\frac{h_m(M_r) + \sum_{i=1}^{n}\sum_{j=1}^{|V_i^r|} L^r_{i,j}}{M_r} \le \frac{I_r + P_r}{M_r} \le C_{\max}$$

For the symmetry consideration on a two-stage hybrid flow shop problem [12], the reduce phase is supposed to process before the map phase. By taking into account the available time I_m and the processing time P_m of the map phase, we have:

$$\frac{h_r(M_m) + \sum_{i=1}^{n}\sum_{j=1}^{|V_i^m|} L^m_{i,j}}{M_m} \le \frac{I_m + P_m}{M_m} \le C_{\max}$$

Therefore,

$$LB_1 = \max\left\{\frac{h_m(M_r)}{M_r} + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^r|} L^r_{i,j}}{M_r}, \frac{h_r(M_m)}{M_m} + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^m|} L^m_{i,j}}{M_m}\right\} \le C_{\max}$$ □

We propose another lower bound LB_2 considering the precedence relation between the two phases (map and reduce).

Definition 2 *Z_i^a is the maximum $L^a_{i,j}$ of all tasks of job J_i at phase a, i.e., $Z_i^a = \max_{j \in V_i^a}\{L^a_{i,j}\}$. Z^a is minimum Z_i^a of all jobs, i.e., $Z^a = \min_{i\in\{1,2,\ldots,n\}}\{Z_i^a\}$, $a \in \{m, r\}$.*

Theorem 2 $LB_2 = \max\left\{Z^m + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^r|} L^r_{i,j}}{M_r},\ Z^r + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^m|} L^m_{i,j}}{M_m}\right\}$ *is a lower bound of the makespan on any feasible solution.*

Proof The reduce phase of a job cannot start until all of its map tasks have been finished. Ideally, there are enough slots for all map tasks to start simultaneously. Then the reduce phase could start only after the map task with the longest processing time is finished, which would lead to the least available time in each reduce slot being Z^m. Therefore, the available setup time of all reduce slots is also at least Z^m. From equation (13),

$Z^m + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^r|} L_{i,j}^r}{M_r} \leq C_{\max}$. The symmetry property of the two-stage hybrid flow shop problem is $Z^r + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^m|} L_{i,j}^m}{M_m} \leq C_{\max}$. Therefore,

$$LB_2 = \max\left\{Z^m + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^r|} L_{i,j}^r}{M_r}, Z^r + \frac{\sum_{i=1}^{n}\sum_{j=1}^{|V_i^m|} L_{i,j}^m}{M_m}\right\} \leq C_{\max} \qquad \square$$

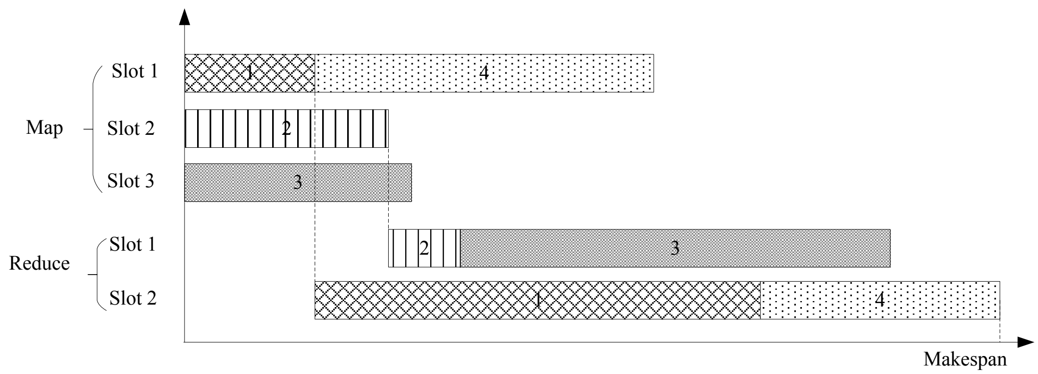

Figure 2: A case for $LB_2 < LB_1$.

$LB_2 > LB_1$ for most cases. However, there are exceptions like the case shown in Figure 2. The cluster has 3 map slots and 2 reduce slots. Four MapReduce jobs need to be processed, each of which has only one map task and one reduce task. According to LB_1, the mean available time of the reduce slots is the average of the processing times of job 1 and job 2 while it is job 1's processing time in terms of LB_2. Obviously, $LB_1 > LB_2$. Therefore, a lower bound of $C_{\max}$ on any feasible solution is $LB = \max\{LB_1,\ LB_2\}$.

4. Heuristics

Three heuristics are proposed for the considered problem in this paper. Generally, heuristics are adopted from those proposed in hybrid flow shop problems, in which jobs are sorted by a sequencing rule at each phase and they are assigned to machines using another rule. The considered problem is unique in that each job consists of multiple tasks, which are the basic units of scheduling. Therefore, three sub-problems should be solved to generate a schedule.

(i) The scheduling sequence of the jobs.

(ii) The task scheduling sequence of each job.

(iii) The task assignment at each phase.

There are many options for sequencing rules and task assignment policies. A heuristic is called job-based if the job sequence is generated priori to the task sequencing for each job. On the contrary, a heuristic is task-based if the job sequence is generated according to the obtained task sequences. In this section, two job-based heuristics and a task-based heuristic are presented.

4.1. Fundamental rules for the three sub-problems

4.1.1. Job sequencing rule

Johnson's algorithm [14] can be used as a job sequencing rule which obtains the optimum of a two-stage flow shop problem minimizing the makespan. Variants of Johnson's algorithm have been applied to many kinds of flow shop problems [38, 17, 13, 22]. However, it is necessary to estimate the duration of each phase before using Johnson's algorithm when there is more than one parallel machine in each phase. Therefore, we need to determine the durations of the map and reduce phases by analyzing processing and setup times of tasks for the problem considered in this paper. Verma et al. [38] calculated the lower bound (Equation(14)) and upper bound (Euqation(15)) of some phase durations for job J_i using the makespan theorem [37]:

$$T_i^{a,low} = \frac{\sum_{j=1}^{|V_i^a|} p_{i,j}^a}{S_i^a} \tag{14}$$

$$T_i^{a,up} = \frac{(\mid V_i^a \mid -1) \cdot \sum_{j=1}^{|V_i^a|} p_{i,j}^a}{S_i^a \cdot \mid V_i^a \mid} + \max_{j \in V_i^a} p_{i,j}^a \tag{15}$$

in which S_a^i is the number of slots allocated to process the tasks of job J_i. We calculate the estimated duration of job J_i at phase a as the weighted sum of the lower bound and upper bound above with weights ω and $1-\omega$, respectively (Equation (16)). $L_{i,j}^a$, the sum of processing and setup times, is regarded as the artificial processing time of task $v_{i,j}^a$. Since the setup time is unknown until the processing slot is determined, we regard the time to read local input data as the setup time.

$$T_i^a = \omega T_i^{a,low} + (1-\omega) T_i^{a,up},\ \omega \in (0,1) \tag{16}$$

With the estimated durations, we use Johnson's algorithm to sort the jobs. The sequencing rule just described is abbreviated to JR_1 which will be used to sort jobs in the map phase in the proposed methods. In order to start the reduce phase as soon as possible, the jobs at the reduce phase are sorted in a non-decreasing order of their completion times at the map phase.

4.1.2. Task sequencing rule

Each stage of the considered hybrid flow shop can be viewed as a parallel machines problem with identical processors ($P||C_{\max}$). The Longest Processing Time first (LPT) rule can obtain a near-optimal solution for problem $P||C_{\max}$ [25]. Therefore, we adopt the LPT rule to sort the tasks of each job at both phases in terms of the artificial processing time $L^a_{i,j}$ of task $v^a_{i,j}$.

4.1.3. Task assignment policies

The most commonly used job-machine assignment policies for traditional hybrid flow shop problems are Earliest Available First (EAF) [25, 15] and Earliest Finishing First (EFF) [17, 13]. EAF results in the least waiting time for jobs while EFF leads jobs to finish as soon as possible. The Latest Available First (LAF) [11] policy is sometimes also adopted. In the MapReduce task scheduling, it is necessary to take into account other factors, such as the load balancing of slots, the data locality of tasks and the precedence constraint between the map and the reduce phases. Since the data locality greatly affects the setup times of tasks, an improvement policy is developed in this paper and discussed in section 4.4.

4.2. Job-based heuristics

This section introduces two job-based scheduling heuristics, EASS (Earliest Available Slot Scheduling) and EFSS (Earliest Finishing Slot Scheduling).

EASS is based on the EAF task assignment policy. As shown in Algorithm 1, EASS first sorts jobs using the JR_1 rule at the map phase and sequences the tasks by the LPT rule. The next available moment for slot k is defined as λ_k, which is initialized as 0. θ^a_i denotes the completion time of phase a of job J_i. At the map phase, the current task is always assigned to the earliest available slot, i.e., the slot $k' = \arg\min_{k\in Q_m} \lambda_k$ is selected, which is implemented by the Task Assignment Procedure (TAP) as shown in Algorithm 2. The completion time of the map phase and the next available moment for slot k' are updated after each assignment. At the reduce phase, the jobs are sorted in non-decreasing order of the completion times at the map phase. Tasks are assigned in the same way at the map phase. To ensure the reduce tasks cannot start until all map tasks complete, the start time of any reduce task is set as no less than the latest completion time of the corresponding job at the map phase (as described by the statement 6 in Algorithm 2). The makespan is the maximum completion time of all jobs at the reduce phase.

Algorithm 1: EASS Heuristic

Input: Job set $\mathbb{J}$, MapReduce cluster Q, processing time $p^a_{i,j}$ of task $v^a_{i,j}$, setup time $s^a_{i,j,k}$ of task $v^a_{i,j}$ processed by slot k.

Output: Makespan of $\mathbb{J}$, $C_{\max}$.

1 **begin**

2   Sort jobs in $\mathbb{J}$ using JR_1 rule `/* Map phase */`

3   **foreach** $k \in Q$ **do**

4     $\lambda_k \leftarrow 0$

5   **foreach** $J_i \in \mathbb{J}$ **do**

6   Sort the tasks in V^m_i and V^r_i respectively using LPT rule

7     $\theta^m_i \leftarrow 0$

8   **foreach** $v^m_{i,j} \in V^m_i$ **do**

9       $k' \leftarrow \arg\min_{k \in Q_m} \lambda_k$

10   Call $TAP(J_i, v^m_{i,j}, \theta^m_i, \lambda_{k'}, m)$

11   Sort jobs in $\mathbb{J}$ in non-decreasing order of θ^m_i `/* Reduce phase */`

12   **foreach** $J_i \in \mathbb{J}$ **do**

13     $\theta^r_i \leftarrow \theta^m_i$

14   **foreach** $v^r_{i,j} \in V^r_i$ **do**

15       $k' \leftarrow \arg\min_{k \in Q_r} \lambda_k$

16   Call $TAP(J_i, v^r_{i,j}, \theta^r_i, \lambda_{k'}, r)$

17   **return** $\max_{i \in \{1,2,\ldots,n\}} \theta^r_i$

EFSS operates in a greedy manner using EFF as the task assignment policy. For each task $v^a_{i,j}$, the slot $k' = \arg\min_{k \in Q_a}\{\lambda_k + s^a_{i,j,k} + p^a_{i,j}\}$ is selected. The procedure of task assignment is also completed by TAP. Jobs and tasks are arranged in the same way as in EASS. EFSS can be obtained from EASS by just changing statement 9 in EASS to $k' = \arg\min_{k \in Q_m}\{\lambda_k + s^m_{i,j,k} + p^m_{i,j}\}$ and the statement 15 to $k' = \arg\min_{k \in Q_r}\{\lambda_k + s^r_{i,j,k} + p^r_{i,j}\}$.

Algorithm 2: TAP (J_i, $v^a_{i,j}$, θ^a_i, $\lambda_{k'}$, a)

begin
- $s^a_{i,j} \leftarrow s^a_{i,j,k'}$
- **if** $a = m$ **then**
  - $c^a_{i,j} \leftarrow \lambda_{k'} + s^a_{i,j} + p^a_{i,j}$
- **else**
  - $c^a_{i,j} \leftarrow \max\{\lambda_{k'}, \theta^m_i\} + s^a_{i,j} + p^a_{i,j}$
- **if** $c^a_{i,j} > \theta^a_i$ **then**
  - $\theta^a_i \leftarrow c^a_{i,j}$
- $\lambda_{k'} \leftarrow c^a_{i,j}$
- **return**

4.3. Task-based heuristic

The proposed job-based methods use the LPT rule to sort tasks only for each job, which could result in non-high quality solutions. A task-based heuristic method, Task-based Scheduling (TBS), is proposed as shown in Algorithm 3. At the map phase, all map tasks are sorted using the LPT rule regardless of the jobs to which they belong. The EFF policy is adopted to assign the tasks to slots. Tasks assigned to each slot are adjusted to make the tasks from the same job adjacent, which ensures that the reduce phase can start as soon as possible once the map phase is finished. Jobs in each slot follow the same order obtained by JR_1 during the adjustment. Since each task stays in the same slot, its setup time remains unchanged before and after the adjustment. The completion time of each task and that of its job at the map phase are updated. Task scheduling in TBS at the reduce phase is similar to that in the EFSS method.

4.4. Improvement policy based on data locality

For each map task, the input data is replicated on different nodes. $g^k_{i,j}$ denotes the input data size of map task $v^m_{i,j}$ on the node to which map slot k belongs and $d^k_{i,j}$ is the size of data read by reduce task $v^r_{i,j}$ from the output data of map task $v^m_{i,k}$. Data transfer time in cluster contains the communication time and the disk I/O operation time. For simplicity, we assume there are three kinds of communication rates among nodes in cluster: non-local rate f_n (network I/O rate among nodes from different rack), rack-local rate f_r

Algorithm 3: TBS Heursitic

Input: Job set $\mathbb{J}$, MapReduce cluster Q, processing time $p^a_{i,j}$ of task $v^a_{i,j}$, setup time $s^a_{i,j,k}$ of task $v^a_{i,j}$ processed by slot k

Output: Makespan of $\mathbb{J}$, $C_{\max}$

begin
    Sort tasks in T_m using LPT rule `/* Map phase */`
    foreach $k \in Q$ **do**
        $\lambda_k \leftarrow 0$
    foreach $J_i \in \mathbb{J}$ **do**
        $\theta^m_i \leftarrow 0$
    foreach $v^m_{i,j} \in T_m$ **do**
        $k' \leftarrow \arg\min_{k \in Q_m} \{\lambda_k + s^m_{i,j,k} + p^m_{i,j}\}$
        Call $TAP(J_i, v^m_{i,j}, \theta^m_i, \lambda_{k'}, m)$
    `/* task moving */`
    foreach *Map slot* $k \in Q_m$ **do**
        Sort the tasks on slot k in the same order of corresponding jobs obtained by JR_1
        Update the completion time of each task and that of its job
    Sort jobs in $\mathbb{J}$ in non-decreasing order of θ^m_i `/* Reduce phase */`
    foreach $J_i \in \mathbb{J}$ **do**
        Sort the tasks in V^r_i using LPT rule
        $\theta^r_i \leftarrow \theta^m_i$
        foreach $v^r_{i,j} \in V^r_i$ **do**
            $k' \leftarrow \arg\min_{k \in Q_r} \{\lambda_k + s^r_{i,j,k} + p^r_{i,j}\}$
            Call $TAP(J_i, v^r_{i,j}, \theta^r_i, \lambda_{k'}, r)$
    return $\max_{i \in \{1,2,\ldots,n\}} \theta^r_i$

(network I/O rate among nodes from the same rack) and node-local rate f_d (disk I/O rate of a local node). The setup time of map task $v^m_{i,j}$ depends on both $g^k_{i,j}$ and the communication rates while that of $v^r_{i,j}$ is determined by $d^k_{i,j}$ and the communication rates.

The schedule-dependent setup times exert a great influence on the scheduling effectiveness. Since the communication rate is one of the crucial factors for setup times, data locality is important in reducing setup times. Actually, three aspects are involved in makespan minimization at the map phase: (i) Assign the map tasks to the nodes with replicas of their input data. (ii) Centralize map tasks of the same job to save communication time. (iii) Decentralize map tasks of different jobs to balance workloads in slots. Relating to the three aspects, we introduce an improvement policy. All replicas of input data are assigned to the nodes using the round-robin way, which balances workloads on the slots, i.e., map tasks of different jobs are decentralized to distinct slots. At the map phase, an attempt to allocate the earliest available slot on the same node to the next task of the current job is made. Although this policy does not lead to the earliest completion of the task, better solution can be obtained because it reserves slots for the successive tasks with local executions. Additionally, the input data placement increases the possibility of the tasks of a job being processed by the same rack.

The improvement policy is applied to EFSS and TBS, the obtained heuristics are called EFSS-L and TBS-L, respectively. Since EASS is based on the EAF task assignment policy, which is the same for the improvement policy, it is unnecessary to construct EASS-L.

4.5. Time complexity of the proposals

In EASS, the time complexity of Step 2 is $O(n \log n)$, that of Step 6 is $O(\sum_{i=1}^{n}(|V_i^m| \log |V_i^m| + |V_i^r| \log |V_i^r|))$, that of the TAP procedure is $O(1)$ while that of Step 9 is $O(\sum_{i=1}^{n} |V_i^m| M_m)$. So the time complexity of the map phase is $O(n \log n + \sum_{i=1}^{n}(|V_i^m| \log |V_i^m| + |V_i^r| \log |V_i^r|) + \sum_{i=1}^{n} |V_i^m| M_m)$ and that of the reduce phase is $O(n \log n + \sum_{i=1}^{n} |V_i^r| M_r)$. Therefore, the time complexity of EASS is $O(n \log n + \sum_{i=1}^{n}[|V_i^m|(\log |V_i^m| + M_m) + |V_i^r|(\log |V_i^r| + M_r)])$.

Since the time complexity of the distinct steps between EASS and EFSS is identical, the time complexity of EFSS is also $O(n \log n + \sum_{i=1}^{n}[|V_i^m|(\log |V_i^m| + M_m) + |V_i^r|(\log |V_i^r| + M_r)])$.

In TBS, the time complexity of the map phase is $O(|T_m| \log |T_m|)$, that is the task moving phase is $O(|T_m| \log |T_m|)$, and that of the reduce phase is $O(n \log n + \sum_{i=1}^{n}[|V_i^r|(\log |V_i^r| + M_r)])$. Therefore, the time complexity of TBS is $O(n \log n + \sum_{i=1}^{n}[|V_i^r|(\log |V_i^r| + M_r)] + |T_m| \log |T_m|)$.

5. Experimental results

This section evaluates the proposed heuristics and improvement policy using realistic workloads derived from the Yahoo! M45 [16] cluster. Job information was randomly generated from data distributions drawn from log files of 10 months. The heuristics were encoded in Java, compiled with Eclipse Helios Release JDK 1.6 and run on a PC with an Intel Core i5-3479 3.7GHz processor with 4GB of RAM.

5.1. Data generation

Jobs and tasks information is based on the analysis performed on a Yahoo! M45 production cluster and was generated as follows [38]: (1) The number of map and reduce tasks for each job was drawn from the normal distributions $N(154,\ 558)$ and $N(19,\ 145)$, respectively. (2) Map and reduce task processing times were generated from the normal distributions $N(50,\ 200)$ and $N(100,\ 300)$, respectively. (3) A bimodal workload was adopted in order to avoid similar job data sets since they were drawn from the same distributions. The data of 80% of the jobs was multiplied by a scale factor uniformly distributed between [1,2] while the rest of the data was scaled using a factor drawn uniformly from [8,10]. After scaling, all data was rounded to the nearest integers.

For cluster setups, the network architecture has a two-level topology in which the rack number is set to 3; the number of nodes takes values from $m \in \{10,\ 15,\ 20,\ 25,\ 30\}$; considering various slot configurations, the number of map slots on each node $ms \in \{2,\ 4,\ 6,\ 8\}$ and that of reduce slots on each node rs is set to 2. Therefore, we obtain four slot ratios: $R_1 = 2:2,\ R_2 = 4:2,\ R_3 = 6:2,\ R_4 = 8:2$. At present, the rate of disk I/O can reach 150 megabytes per second (MB/s). We set f_d to 100 MB/s in view of an average sense of I/O performance. Gigabit Ethernet is a common option for Hadoop clusters with a maximum communication rate exceeding 100 MB/s. Moreover, the aggregate bandwidth between nodes on the same rack is much greater than that between nodes on different racks [40]. With these factors, we set $f_r = 50$ MB/s and $f_n = 30$ MB/s, respectively.

The input data size of each map task is drawn from $\{128,\ 192,\ 256,\ 320\}$ with the unit being MB. As a result, the setup time of the map task is approximately between 1 and 11 seconds. For the map task, the ratio of the time for reading input data to the processing time is about 1:10 [3]. According to the data distribution of the processing time, it is reasonable to set the average setup time of the map task as 5 seconds. The replica number of data blocks on HDFS (Hadoop Distributed File System) is 4 and the replicas of each task are placed on 4 consecutive nodes in a round-robin way. Each map task is assumed to have only one block of input data. For simplicity, the amount of input data D_{in} is linear to that of the output data D_{out} at the map phase, i.e., $D_{out} = \sigma D_{in}$. In a MapReduce production cluster, most jobs are data-aggregate or data-transform with a $\sigma \leq 1$ [7]. σ was randomly generated from $\{0.2,\ 0.4,\ 0.6,\ 0.8,\ 1.0\}$ to reflect different types of jobs. 30 instances are randomly generated for each job number $n \in \{50,\ 100,\ 150,\ 200,\ 250\}$, i.e., there are $5 \times 30 = 150$ instances in total. Parameters of each instance contain the number of map tasks, the number of reduce tasks, σ and processing times of its tasks.

The lower bound proposed in section 3 is used to evaluate the methods. Relative Error (RE) [13] is defined as:

$$RE = \frac{C_{\max} - LB}{LB} \times 100\% \tag{17}$$

Smaller RE values suggest better solutions as the obtained makespan is closer to the lower bound.

5.2. Results

5.2.1. Parameter calibration

Equation (16) implies that the parameter ω determines the final estimated durations of the jobs based on the estimated phase (map/reduce) length. Different job sequences could be generated by the same methods for distinct ω, i.e., ω is crucial for the performance of the methods. Furthermore, the estimated phase durations depend on the number of slots at the map/reduce phase. Therefore, we need to set a good ω value for the proposed heuristics. To determine the appropriate value of ω, we tested EASS, EFSS and TBS over different values of ω $(0.1,\ 0.3,\ 0.5,\ 0.7,\ 0.9)$. Four slot ratios ($R_1$, R_2, R_3, R_4, the number of map slots to that of reduce slots) are tested. For each ratio and ω, we also test 5 values for the number of nodes $(10, 15, 20, 25, 30)$. All these factors are controlled in an experimental design so there are $3 \times 5 \times 4 \times 5 = 300$ treatments. We also control the number of jobs n as an instance factor, with the aforementioned 5 levels, which increases the number of treatments to 1500. The 30 random instances for each level of n are tested so the total number of results in the calibration experiment is 45000. All this data is fed to the Analysis of Variance technique (ANOVA). The response variable in the experiment is the RE for each algorithm in each instance. ANOVA is a very robust parametric procedure and there are a number of hypotheses that should be ideally met by the experimental data. Among these, the main three are (in order of importance): independence of the residuals, homoscedasticity or homogeneity of the factor's levels variance and normality in the residuals of the model. Apart from a slight non-normality in the residuals, we can accept all hypotheses easily. Note that despite being a parametric test, ANOVA has been demonstrated to be really robust. For example [28] applied ANOVA to data that severely violated normality and tested it together with other non-parametric tools. The conclusions were that ANOVA is largely unaffected by this lack of normality and due to its additional statistical power, it is a much more preferable technique. Additionally, as explained in [2] and in greater detail in [31], computer experimentation is a controllable environment where few things can go wrong as regards the ANOVA.

All studied factors in the ANOVA resulted in being statistically significant with p-values very close to zero. The most insightful result of the ANOVA is a means plot with an additional statistical test to check which averages of the levels and variants of the factors that have been proved to be statistically significant are indeed different from each other. The means plot with 95% confidence level Tukey's Honest Significance Differences (HSD) intervals for the interaction between ω and algorithms is shown in Figure 3. Non-overlapping confidence intervals between any two pairs of plotted averages imply that the observed differences in such averages are statistically significant at the indicated confidence level.

From the result of the calibration experiment, it is clear that different ω values have an effect on all three algorithms. All other controlled factors also influence the RE response variable, especially the slot ratios but they do not strongly interact with ω meaning that this factor is robust and the best level is 0.7 for all algorithms and slot ratios. Of course, we could dig deeper and set specific values of ω for all combinations of instance factors but this would result in an overcalibration. It is simpler and fairer to set the same ω value for all instances. It has to be noted that in most cases the value of RE is low, especially when the number of jobs is large and for ratios R_1 and R_2. This is a very good result since at the same time it empirically demonstrates the tightness of the proposed bounds and the effectiveness of the presented heuristics. However, when the number of jobs is equal to 20 and particularly for ratios R_3 and R_4, the RE values surpass 15% in some cases. This could be due to the bound not being so tight and/or the proposed methods not

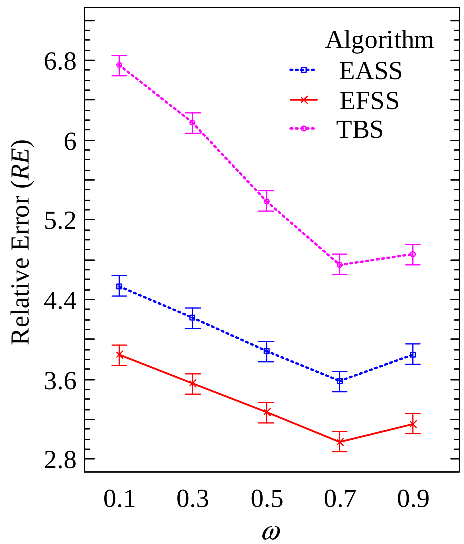

Figure 3: Means plot of the Relative Error (ER) and 95% confidence level Tukey's HSD intervals for the interaction between the factor ω and the type of algorithm.

giving such good solutions.

5.2.2. Comparison results

We now test the three proposed heuristics (EASS, EFSS, TBS) and the two with the improvement policy (EFSS-L, TBS-L). We test all these 5 algorithms again with the previous 150 instances (30 replicates for each job number n). Also, the previous four slot ratios and five number of nodes are tested. The total number of results this time is $5 \times 4 \times 5 \times 150 = 15000$. Note that to avoid bias in the result, we have not taken the results from the previous calibration experiment but rather we have run all methods again. Recall that from the result of the calibration, ω is set to 0.7 in all final experiments. Several of the tested factors have an effect on the performance of the methods. Therefore, we compare the heuristics in different scenarios. First we report the average RE results of each algorithm as a function of the slot ratio, number of nodes and job number n in Table 2. Later we will analyze the statistical significance of the differences in the observed averages. The CPU times employed by the proposed algorithms depend mainly on the number of jobs n. This information, together with the global average RE values is synthesized in Table 3.

Table 2: Average Relative Error (RE) for the proposed heuristics as a function of the slot ratio, node number and number of jobs n. Each cell inside the table is the average RE for the 30 tested instances.

		Slot ratio																			
		R_1					R_2					R_3					R_4				
		Node number					Node number					Node number					Node number				
Algorithm	n	10	15	20	25	30	10	15	20	25	30	10	15	20	25	30	10	15	20	25	30
EASS	50	1.30	2.02	2.64	2.78	3.11	1.52	1.89	2.18	2.60	3.12	2.17	2.68	3.53	5.50	10.39	7.50	8.45	10.79	14.60	19.87
	100	1.20	1.89	2.46	2.51	2.75	1.31	1.65	1.85	2.09	2.48	1.65	2.02	2.29	2.41	2.80	6.74	7.19	7.67	8.71	10.86
	150	1.17	1.85	2.41	2.49	2.73	1.22	1.49	1.66	1.94	2.33	1.34	1.66	1.84	2.01	2.17	7.03	7.31	7.68	8.31	9.55
	200	1.12	1.79	2.34	2.41	2.63	1.15	1.41	1.57	1.84	2.19	1.23	1.51	1.71	1.82	1.93	6.69	6.95	7.08	7.55	8.40
	250	1.13	1.80	2.35	2.42	2.64	1.14	1.41	1.56	1.83	2.18	1.21	1.50	1.67	1.81	1.91	6.83	7.06	7.20	7.44	7.88
Average		1.18	1.87	2.44	2.52	2.77	1.27	1.57	1.76	2.06	2.46	1.52	1.87	2.21	2.71	3.84	6.96	7.39	8.08	9.32	11.31
EFSS	50	0.58	0.81	1.10	1.18	1.35	0.71	0.93	1.15	1.33	1.59	1.32	1.59	2.16	3.49	6.68	7.21	8.28	11.78	15.12	20.37
	100	0.51	0.73	0.99	1.00	1.11	0.57	0.77	0.94	0.97	1.13	0.89	1.12	1.38	1.43	1.65	6.64	7.15	7.76	10.02	12.43
	150	0.47	0.67	0.92	0.94	1.08	0.47	0.63	0.78	0.86	1.00	0.56	0.74	0.93	1.01	1.14	6.95	7.26	7.68	8.63	10.73
	200	0.45	0.66	0.90	0.92	1.02	0.43	0.58	0.73	0.80	0.91	0.46	0.61	0.82	0.87	0.96	6.63	6.90	7.02	7.71	9.17
	250	0.45	0.66	0.91	0.91	1.02	0.43	0.58	0.73	0.79	0.90	0.44	0.60	0.79	0.86	0.93	6.79	7.02	7.17	7.42	8.17
Average		0.49	0.71	0.96	0.99	1.12	0.52	0.70	0.87	0.95	1.10	0.73	0.93	1.22	1.53	2.27	6.84	7.32	8.28	9.78	12.17
EFSS-L	50	0.19	0.24	1.82	0.38	0.47	0.40	0.50	2.13	0.80	0.98	1.07	1.29	2.97	3.25	7.51	7.21	8.42	11.52	16.17	20.55
	100	0.12	0.15	1.85	0.19	0.21	0.24	0.33	2.05	0.40	0.47	0.63	0.77	2.41	0.98	1.27	6.64	7.15	7.76	10.15	12.55
	150	0.07	0.08	1.43	0.11	0.14	0.14	0.18	1.53	0.25	0.32	0.28	0.36	1.69	0.50	0.59	6.96	7.27	7.71	8.72	10.70
	200	0.05	0.07	1.51	0.10	0.10	0.11	0.13	1.59	0.20	0.22	0.17	0.22	1.70	0.33	0.38	6.63	6.91	7.05	7.73	9.15
	250	0.05	0.06	1.40	0.08	0.09	0.10	0.13	1.47	0.18	0.21	0.15	0.20	1.56	0.31	0.35	6.79	7.03	7.17	7.43	8.22
Average		0.10	0.12	1.60	0.17	0.20	0.20	0.26	1.75	0.37	0.44	0.46	0.57	2.06	1.07	2.02	6.85	7.35	8.24	10.04	12.23
TBS	50	0.76	1.02	1.31	1.34	1.48	0.90	1.26	1.64	2.02	2.41	4.62	8.93	13.54	18.63	23.94	11.35	15.86	21.96	29.35	35.68
	100	0.73	1.01	1.23	1.20	1.32	0.71	0.96	1.15	1.20	1.43	1.40	2.30	3.84	5.90	9.51	7.41	8.83	10.89	15.60	19.12
	150	0.73	1.02	1.24	1.22	1.31	0.62	0.82	0.95	1.01	1.13	0.75	1.11	1.51	2.44	4.09	7.55	8.18	9.04	10.75	13.06
	200	0.73	1.02	1.25	1.22	1.30	0.58	0.75	0.87	0.91	1.00	0.58	0.80	1.04	1.38	1.99	7.02	7.52	7.80	8.79	10.57
	250	0.76	1.05	1.28	1.27	1.35	0.59	0.73	0.84	0.90	0.97	0.54	0.73	0.91	1.07	1.29	7.13	7.55	7.90	8.36	9.31
Average		0.74	1.02	1.26	1.25	1.35	0.68	0.90	1.09	1.21	1.39	1.58	2.77	4.17	5.88	8.16	8.10	9.59	11.52	14.57	17.55
TBS-L	50	0.17	0.25	1.84	0.40	0.54	0.54	0.80	2.54	1.57	1.99	4.01	8.37	13.09	17.68	23.22	11.02	15.19	21.99	28.06	35.68
	100	0.11	0.16	1.87	0.22	0.29	0.30	0.45	2.16	0.69	0.84	1.26	2.14	4.36	5.50	8.55	7.59	8.95	11.20	15.11	18.51
	150	0.06	0.09	1.44	0.15	0.18	0.19	0.28	1.65	0.43	0.48	0.42	0.63	2.21	1.96	4.03	7.51	8.21	9.06	10.39	13.30
	200	0.05	0.06	1.51	0.10	0.14	0.12	0.20	1.65	0.31	0.38	0.24	0.41	1.83	0.92	1.58	7.05	7.48	7.83	8.52	10.20
	250	0.04	0.06	1.39	0.09	0.12	0.10	0.15	1.52	0.27	0.30	0.19	0.30	1.64	0.53	0.99	7.12	7.50	7.81	8.31	9.15
Average		0.09	0.12	1.61	0.19	0.25	0.25	0.38	1.90	0.65	0.80	1.22	2.37	4.63	5.32	7.67	8.06	9.47	11.58	14.08	17.37
Average		0.52	0.77	1.57	1.03	1.14	0.58	0.76	1.48	1.05	1.24	1.10	1.70	2.86	3.30	4.79	7.36	8.22	9.54	11.56	14.13

Table 3: Average RE and CPU times (in milliseconds) for the proposed heuristics as a function of the number of jobs n.

	EASS		EFSS		EFSS-L		TBS		TBS-L	
n	RE	Time	RE	Time	RE	Time	RE	Time	RE	Time
50	5.43	23.12	4.44	634.90	4.39	597.80	9.90	606.08	9.45	602.06
100	3.63	47.82	2.96	1232.31	2.82	1183.88	4.79	1200.36	4.51	1189.97
150	3.41	71.93	2.67	1858.74	2.45	1793.58	3.43	1818.60	3.13	1808.93
200	3.17	104.56	2.43	2462.68	2.22	2335.66	2.86	2418.81	2.53	2393.35
250	3.15	124.62	2.38	3034.95	2.15	2928.35	2.73	3003.98	2.38	2961.86
Average	3.76	74.41	2.97	1844.72	2.81	1767.85	4.74	1809.57	4.40	1791.23

We comment on the main findings below.

(i) Slot ratios. For the slot ratio $R_1 = 2:2$, EFSS-L and TBS-L outperform the other methods; when the ratio becomes $R_2 = 4:2$, EFSS-L is the best method but differences are small between EFSS and TBS-L. For $R_3 = 6:2$ EFSS and EFSS-L are the best. Lastly, for $R_4 = 8:2$ all methods except TBS and TBS-L show comparable performance. The methods adopting the improvement policy perform, on average, better than those without the policy.

(ii) Slot ratio and the number of jobs n. The average RE for each job size increases with the slot ratio. Additionally, for each slot ratio, the average RE shows a decreasing trend as n increases. The trend is much more notable when n changes from 50 to 100 and not so obvious after n reaches 200. We can argue that all methods perform better for larger job sizes which are independent from the slot ratio.

(iii) Job size n. EFSS-L outperforms the other methods for all job sizes. The performance of TBS and TBS-L is much worse than the other methods when job sizes are no more than 100. However, they perform much better as the job size increases which indicates that these methods are effective for a large number of jobs. We can conclude that EFSS-L is good for different cluster setups and job sizes. Similarly, the CPU times of the compared methods except EASS are similar for each n. Though there are significant differences between the CPU times of EASS and those of EFSS, their time complexities are identical. In fact, the ratio of the CPU time of EFSS to that of EASS is almost a constant, about 22 for each n, which implies the same time complexity.

(iv) Node size. We focus on the case when the map/reduce slot ratio is $R_2 = 4 : 2$. We can see from Table 3 that EFSS-L achieves the least average RE for almost all combinations of job and node sizes except the case $m = 20$. EFSS-L and TBS-L outperform EFSS and TBS, supporting the effectiveness of the proposed improvement policy. Moreover, all methods show a trend of worse performance as the node size increases except when $m = 20$ with a fixed job size. For any node size, all methods tend to perform better with more jobs. We can argue that in most cases, the proposed methods are appropriate for resource-constrained (fewer nodes but more jobs) environments. For the case when the node number is 20, we analyze and propose possible causes. As for the results of the rest of the slot ratios, it is observed that similar patterns exist for the differences in performance with more jobs and nodes, which are not shown due to space limitations.

(v) Input data distribution. The performance of the different methods shows that the improvement policy is far less effective when the node size is 20. A possible reason is that data locality for tasks selecting slots would lead to unbalanced workload in the slots if the input data is distributed unevenly, which would result in worse makespans. In normal cases, there is some common input data among the set of nodes holding the replicas of input data of different tasks which balance the workload of each slot. However, a special case is that the overlapping becomes less important when the node size is a multiple of the replica number if all the replicas are placed on HDFS in a round-robin way. For example, the node number 20 is a multiple of the replica number 4, the improvement policy fails in this special case. The above analysis is verified by experiments with the configured parameters except the node size m, which takes values from $\{12, 16, 20, 24, 28\}$. The experimental results show that EFSS achieves the best performance, EFSS-L and TBS-L get worse RE than EFSS and TBS in almost all cases. The two methods with the improvement policy are even outperformed by EASS in the worst case. This result supports the earlier analysis and indicates that the distribution of input data may greatly affect the proposed improvement policy. Therefore, it is necessary to fully take into account the specific configurations of a cluster when selecting methods for scheduling.

Now we proceed to the statistical analysis of the experimental results given in Table 2. While there are large differences in the observed averages, we still need to check if these differences are indeed statistically significant. We use the same ANOVA tool as before, with the same factors and response variables as used in the experiment.

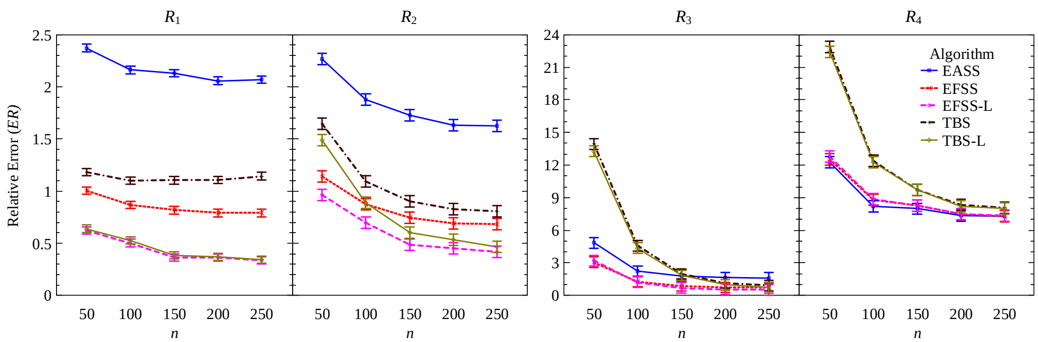

Figure 4: Means plot of the Relative Error (ER) and 95% confidence level Tukey's HSD intervals for the interaction between slot ratio, the number of jobs n and the type of algorithm.

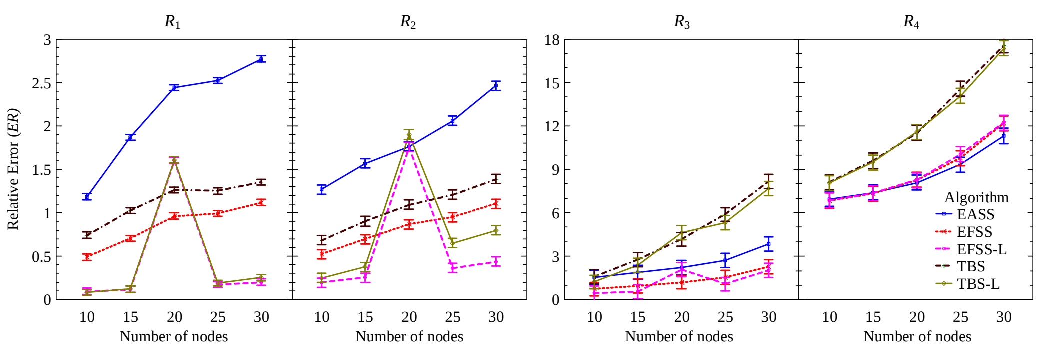

Figure 5: Means plot of the Relative Error (ER) and 95% confidence level Tukey's HSD intervals for the interaction between slot ratio, the number of nodes and the type of algorithm.

As can be seen, the performance of the proposed methods is largely affected, in a sound and statistical way, by the slot ratios. Additionally, the number of jobs n and the number of nodes also affect algorithms in a significant manner. Overall, when differences between the averages reported in Table 2 are small between any algorithm and considered factor, they end up not being statistically significant. Only large differences can be generalized over other workloads (inference to the universe of potential instances). In summary, all tables and figures support the idea that the effectiveness of the proposed methods decreases with the job size n. Bigger job sizes n imply lower RE values, i.e., a greater number of jobs involved in the MapReduce computing framework leads to the makespan being closer to the lower bound. Therefore, the proposed methods are suitable for large-scale data processing systems. Another plausible explanation is that the proposed bounds are weaker for smaller job sizes and therefore the calculated RE values are affected. Considering the difficulty of the proposed model, it is not possible to solve even the smallest considered instances of 50 jobs optimally so it is not possible to check the tightness of the bound.

6. Conclusions and future research

In this paper, the scheduling problem of periodical batch jobs in MapReduce clusters with makespan minimization is considered. The problem is modeled as a general two-stage hybrid flow shop scheduling problem with schedule-dependent setup times and multiple tasks per job at each stage. A tight lower bound of the makespan is derived. Three heuristics EASS, EFSS and TBS are developed to solve the problem and an improvement policy based on data locality is presented to enhance the methods. Computational results have shown that the performance of the different methods highly depends on the number of jobs and cluster setups (map/reduce slot number ratio and node size). The effectiveness of the improvement policy is carefully tested indicating that EFSS-L is the best method in most cases. Finally, we have analyzed the special case when the improvement policy fails, for which an additional experiment is conducted to examine the analysis. In other words, the distribution of input data may affect the effectiveness of methods.

Future research directions involve the impact of more cluster setups on method performance such as the number of racks, number of data replicas, network topology and more extensive map/reduce slot number ratios, etc. Other promising research avenues involve more practical modeling of the scheduling problem considered. For example, the reduce phase can start as soon as one of the map tasks is completed in real MapReduce implementations rather than after the whole map phase. The time when the reduce phase is allowed to start is configurable and thus able to be incorporated in the model.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61272377) and the Specialized Research Fund for the Doctoral Program of Higher Education (No. 20120092110027). Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "RESULT - Realistic Extended Scheduling Using Light Techniques" (No. DPI2012-36243-C02-01) partially financed with FEDER funds.

References

[1] Asahara, M., Nakadai, S., Araki, T., 2012. LoadAtomizer: A locality and I/O load aware task scheduler for mapreduce. In: Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, pp. 317–324.

[2] Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (Eds.), 2010. Experimental Methods for the Analysis of Optimization Algorithms. Springer, New York.

[3] Berlińska, J., Drozdowski, M., 2011. Scheduling divisible MapReduce computations. Journal of Parallel and Distributed Computing 71 (3), 450–459.

[4] Brucker, P., 2007. Scheduling algorithms, 5th Edition. Vol. 3. Springer.

[5] Chang, H., Kodialam, M., Kompella, R. R., Lakshman, T., Lee, M., Mukherjee, S., 2011. Scheduling in MapReduce-like systems for fast completion time. In: INFOCOM, 2011 Proceedings IEEE. IEEE, pp. 3074–3082.

[6] Chen, Q., Zhang, D., Guo, M., Deng, Q., Guo, S., 2010. SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In: Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on. IEEE, pp. 2736–2743.

[7] Chen, Y., Ganapathi, A., Griffith, R., Katz, R., 2011. The case for evaluating MapReduce performance using workload suites. In: Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on. IEEE, pp. 390–399.

[8] Czyżewski, A., Bratoszewski, P., Ciarkowski, A., Cichowski, J., Lisowski, K., Szczodrak, M., Szwoch, G., Krawczyk, H., 2015. Massive surveillance data processing with supercomputing cluster. Information Sciences 296, 322–344.

[9] Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM 51 (1), 107–113.

[10] Fischer, M. J., Su, X., Yin, Y., 2010. Assigning tasks for efficiency in Hadoop. In: Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures. ACM, pp. 30–39.

[11] Gupta, J. N. D., 1988. Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society 39 (4), 359–364.

[12] Haouari, M., M'Hallah, R., 1997. Heuristic algorithms for the two-stage hybrid flowshop problem. Operations Research Letters 21 (1), 43–53.

[13] Huang, W., Li, S., 1998. A two-stage hybrid flowshop with uniform machines and setup times. Mathematical and Computer Modelling 27 (2), 27–45.

[14] Johnson, S. M., 1954. Optimal two-and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1 (1), 61–68.

[15] Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., Werner, F., 2008. Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. The International Journal of Advanced Manufacturing Technology 37 (3-4), 354–370.

[16] Kavulya, S., Tan, J., Gandhi, R., Narasimhan, P., 2010. An analysis of traces from a production MapReduce cluster. In: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on. IEEE, pp. 94–103.

[17] Kurz, M. E., Askin, R. G., 2004. Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research 159 (1), 66–82.

[18] Lee, C.-Y., Vairaktarakis, G. L., 1994. Minimizing makespan in hybrid flowshops. Operations Research Letters 16 (3), 149–158.

[19] Lu, P., Lee, Y. C., Wang, C., Zhou, B. B., Chen, J., Zomaya, A. Y., 2012. Workload characteristic oriented scheduler for MapReduce. In: Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems. IEEE Computer Society, pp. 156–163.

[20] Mika, M., Waligóra, G., Węglarz, J., 2008. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research 187 (3), 1238–1250.

[21] Moseley, B., Dasgupta, A., Kumar, R., Sarlós, T., 2011. On scheduling in Map-Reduce and flow-shops. In: Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures. ACM, pp. 289–298.

[22] Oğuz, C., Fikret Ercan, M., Edwin Cheng, T. C., Fung, Y.-F., 2003. Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. European Journal of Operational Research 149 (2), 390–403.

[23] Phan, L. T., Zhang, Z., Loo, B. T., Lee, I., 2010. Real-time MapReduce scheduling. Tech. Report, UPenn.

[24] Philip Chen, C., Zhang, C.-Y., 2014. Data-intensive applications, challenges, techniques and technologies: A survey on big data. Information Sciences 275, 314–347.

[25] Pinedo, M., 2012. Scheduling: theory, algorithms, and systems, 4th Edition. Springer.

[26] Polo, J., Carrera, D., Becerra, Y., Torres, J., Ayguadé, E., Steinder, M., Whalley, I., 2010. Performance-driven task co-scheduling for MapReduce environments. In: Network Operations and Management Symposium (NOMS), 2010 IEEE. IEEE, pp. 373–380.

[27] Radenski, A., Ehwerhemuepha, L., 2014. Speeding-up codon analysis on the cloud with local mapreduce aggregation. Information Sciences 263, 175–185.

[28] Rasch, D., Guiard, V., 2004. The robustness of parametric statistical methods. Psychology Science 46 (2), 175–208.

[29] Riane, F., Artiba, A., E. Elmaghraby, S., 1998. A hybrid three-stage flowshop problem: efficient heuristics to minimize makespan. European Journal of Operational Research 109 (2), 321–329.

[30] Ribas, I., Leisten, R., Framinan, J. M., 2010. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research 37 (8), 1439–1454.

[31] Ridge, E., Kudenko, D., 2010. Tuning an algorithm using design of experiments. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (Eds.), Experimental Methods for the Analysis of Optimization Algorithms. Springer, New York, Ch. 11, pp. 265–286.

[32] Ruiz, R., Vázquez-Rodríguez, J. A., 2010. The hybrid flow shop scheduling problem. European Journal of Operational Research 205 (1), 1–18.

[33] Shih, H.-Y., Huang, J.-J., Leu, J.-S., 2012. Dynamic slot-based task scheduling based on node workload in a MapReducecomputation model. In: Anti-Counterfeiting, Security and Identification (ASID), 2012 International Conference on. IEEE, pp. 1–5.

[34] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R., Liu, H., 2010. Data warehousing and analytics infrastructure at facebook. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, pp. 1013–1020.

[35] Tian, C., Zhou, H., He, Y., Zha, L., 2009. A dynamic MapReduce scheduler for heterogeneous workloads. In: Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on. IEEE, pp. 218–224.

[36] Verma, A., Cherkasova, L., Campbell, R. H., 2011. Play it again, SimMR! In: Cluster Computing (CLUSTER), 2011 IEEE International Conference on. IEEE, pp. 253–261.

[37] Verma, A., Cherkasova, L., Campbell, R. H., 2011. Resource provisioning framework for MapReduce jobs with performance goals. In: Kon, F., Kermarrec, A.-M. (Eds.), Lecture Notes in Computer Science. Vol. 7049. Springer, pp. 165–186.

[38] Verma, A., Cherkasova, L., Campbell, R. H., 2013. Orchestrating an ensemble of MapReduce jobs for minimizing their makespan. IEEE Transactions on Dependable and Secure Computing 10 (5), 314–327.

[39] Wang, Y., Shi, W., 2014. Budget-driven scheduling algorithms for batches of mapreduce jobs in heterogeneous clouds. IEEE Transactions on Cloud Computing 2 (3), 306–319.

[40] White, T., 2009. Hadoop: The Definitive Guide. O'Reilly Media.

[41] Wolf, J., Rajan, D., Hildrum, K., Khandekar, R., Kumar, V., Parekh, S., Wu, K.-L., Balmin, A., 2010. FLEX: A slot allocation scheduling optimizer for MapReduce workloads. In: Middleware 2010. Springer, pp. 1–20.

[42] Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S., Stoica, I., 2009. Job scheduling for multi-user MapReduce clusters. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55.

[43] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I., 2010. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In: EuroSys'10 - Proceedings of the EuroSys 2010 Conference. pp. 265–278.

[44] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., Stoica, I., 2008. Improving MapReduce performance in heterogeneous environments. In: OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation. pp. 29–42.