J Supercomput
DOI 10.1007/s11227-016-1737-4

HScheduler: an optimal approach to minimize the makespan of multiple MapReduce jobs

Wenhong Tian[1,3,4] · Guozhong Li[1] · Wutong Yang[1] · Rajkumar Buyya[2]

Abstract Large-scale MapReduce clusters that routinely process big data bring challenges to the cloud computing. One of the key challenges is to reduce the response time of these MapReduce clusters by minimizing their makespans. It is observed that the order in which these jobs are executed can have a significant impact on their overall makespans and resource utilization. In this work, we consider a scheduling model for multiple MapReduce jobs. The goal is to design a job scheduler that minimizes the makespan of such a set of MapReduce jobs. We exploit classical Johnson model and propose a novel framework HScheduler, which combines features of both classical Johnson's algorithm and MapReduce to minimize the makespan for both offline and online jobs. Our Offline HScheduler reaches the theoretical lower bound (optimum) and Online HScheduler is 2-competitive which is the best-known constant ratio for minimizing the makespan. Through extensive real data tests, we find that HScheduler has better performance than the best-known approach by 10.6–11.7 % on average for offline scheduling and 8–10 % on average for online scheduling. The HScheduler can be applied to improve responsive time, throughput and energy efficiency in cloud computing.

This research is sponsored by the Natural Science Foundation of China (NSFC) Grant 61450110440.

✉ Wenhong Tian
tian_wenhong@uestc.edu.cn

1 School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu, China

2 The University of Melbourne, Parkville, Australia

3 Big Data Research Center at University of Electronic Science and Technology of China, (UESTC), Chengdu, China

4 Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China

Published online: 04 May 2016

Keywords Hadoop · MapReduce · Batch workloads · Optimized schedule · Minimized makespan

1 Introduction

With the rapid increase in size and number of jobs that are being processed in the MapReduce framework, efficiently scheduling multiple jobs under this framework is becoming increasingly important. Job scheduling in MapReduce framework brings a new challenge to Cloud computing [1] such as minimizing the makespan, load balancing and reducing data skew. Originally, Hadoop was designed for periodically running large batch workloads with a First-In-First-Out (FIFO) scheduler. As the number of users sharing the same MapReduce cluster increased, there are Capacity scheduler [2] and Hadoop Fair Scheduler (HFS) [3] which intend to support more efficient cluster sharing. There are also a few research prototypes of Hadoop schedulers that aim to optimize explicitly some given scheduling metrics, e.g., FLEX [4], ARIA [5]. A MapReduce simulator called SimMR [6] is also developed to simulate different workloads and performances of MapReduce. Yao et al. [7] proposed a scheme which uses slot ratio between Map and Reduce tasks as a tunable knob for dynamically allocating slots. However, as pointed out in [1], the existing schedulers do not provide a support for minimizing the makespan for a set of jobs.

Starfish project [8] proposes a workflow-aware scheduler that correlates data (block) placement with task scheduling to optimize the workflow completion time. Moseley et al. [9] formulate MapReduce scheduling as a generalized version of the classical two-stage flexible flow-shop problem with identical machines; they provide a 12-approximation algorithm for the offline problem of minimizing the total flow time, which is the sum of the time between the arrival and the completion of each job. Zhu et al. [7] consider non-preemptive case to propose $\frac{3}{2}$-approximation for offline scheduling regarding the makespan. In [1,10], the authors propose heuristics to minimize the makespan, the proposed algorithm called BalancedPools by considering two pools for a Hadoop cluster. This work is closely related to our research in that both are based on Johnson's model and minimizing the makespan. However, our present work modifies Johnson model and provides optimal solution to offline scheduling and 2-competitive solution to online scheduling while Verma et al. [10] did not modify Johnson's model and provided separating pools (called BalancedPools) for minimizing the makespan. BalancedPools is a heuristic approach but not optimal in many cases, and there is still room for improving the performance of MapReduce regarding minimizing the makespan.

As for online scheduling, Zheng et al. [11] propose a new analytical technique for MapReduce schedulers by minimizing the total flow time of online jobs and claim that no online algorithm can achieve a constant competitive ratio for non-preemptive tasks, and provide a slightly weaker metric of performance called the efficiency ratio for evaluation.

In summary, there are only a small number of online scheduling algorithms in open literature and still much room for improving the performance of MapReduce regarding minimizing the makespan. Therefore, we propose new modeling and scheduling

approaches for both offline and online jobs in the following sections. The major contributions of this paper include:

(1) provided a new modeling and scheduling approach for multiple MapReduce jobs;
(2) proposed an optimal algorithm for offline scheduling considering Map and Reduce phases by adapting classical Johnson's model;
(3) the proposed online scheduling is 2-competitive, which is the best-known approximation for online scheduling of multiple MapReduce jobs regarding minimizing the makespan.
(4) validating both offline and online algorithms through real-data tests and simulation.

2 Problem formulation

A MapReduce performance model is introduced in [1,5,10]. The model can be used for predicting the completion time of the Map and Reduce stages as a function of the input dataset size and allocated resources. *In this paper, we consider Map and Reduce two stages where Map stage includes time for job set-up, splitting, mapping to produce (key, value) pairs and Reduce stage includes time for combining, sorting, shuffle, generating final outputs. Same as in* [1], *we define the job execution time as the sum of the complementary, non-overlapping map and reduce stage execution times, i.e., we represent the duration of the first shuffle using a fraction of its non-overlapping time with the duration of the map stage. So that for each job, its Map stage and Reduce stage are not overlapped.*

Definition 1 MapReduce slots. Depending on the configuration of a Hadoop cluster, each node in the cluster can proceed $P_m^{\max}$ map and $P_r^{\max}$ reduce tasks simultaneously. This Hadoop cluster is called having $P_m^{\max} \times P_r^{\max}$ MapReduce slots.

Definition 2 Execution waves. If the required number of MapReduce slots of a task is greater than the number of MapReduce slots available in the cluster, the task assignment proceeds in multiple rounds, each round is called an execution wave.

Definition 3 The makespan of a set of MapReduce jobs is the total time length (the difference) between the end-time of the last scheduled MapReduce job and the start-time of the first scheduled MapReduce job. We denote the makespan as $C_{\max}$.

Figure 1 shows an example executed in two waves of 20×20 MapReduce slots. Consider a job that is represented as a set of n tasks processed in Hadoop environments. Each MapReduce job consists of a specified number of map and reduce tasks. The job execution time and specifications of the execution depend on the amount of resources (map and reduce slots) allocated to the job. A simple abstraction is adopted [1], where each MapReduce job j_i is defined by durations of its Map and Reduce stages m_i and r_i, respectively, i.e., $j_i = (m_i, r_i)$. Let us consider the execution of two independent MapReduce jobs j_1 and j_2 in a Hadoop cluster with an FIFO scheduler. There are no data dependencies between these jobs. Therefore, once the first job completes its Map stage and begins Reduce stage processing, the next job can start its map stage execution

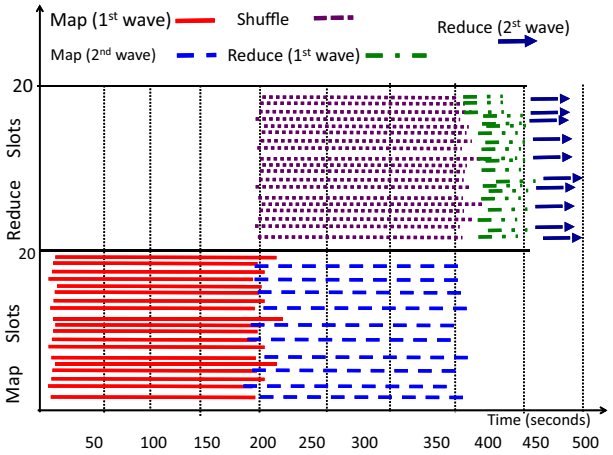

Fig. 1 An execution example of Terasort [12] in a 20 × 20 MapReduce slots

with the released map resources in a pipelined fashion. There may be an "overlap" in executions of map stage of the next job and the reduce stage of the previous one.

We further consider the following problem. Let $J = \{j_1, j_2, \ldots, j_n\}$ be a set of n MapReduce jobs with no data dependencies between them. j_i requests $R_i^m \times R_i^r$ MapReduce slots and has Map and Reduce phase durations (m_i, r_i), respectively. The system scheduler can change a job's MapReduce slot allocation depending on available resources. Let $C_{\max}$ be the makespan of all n jobs. For this, we aim to determine an order (a schedule ϕ) of execution of jobs $j_i \in J$ such that the makespan of all jobs is minimized. Let us set the end-time of Map stage and start-time of Reduce stage of job j_i as (t_m^i, t_r^i), respectively; and actually allocated MapReduce slots for job j_i is $P_m^i \times P_r^i$. The max available MapReduce slots in the Hadoop cluster is $P_m^{\max} \times P_r^{\max}$. Formally, the problem of minimizing the makespan $(C_{\max})$ therefore can be formulated as:

$$\text{Min} \quad C_{\max} \tag{1}$$

$$\text{subject to:} \quad \forall\, j_i, \quad P_m^i \le P_m^{\max}, \quad \text{and} \quad P_r^i \le P_r^{\max}, \tag{2}$$

$$\forall\, j_i, \quad t_r^i \ge t_m^i, \quad \text{and} \tag{3}$$

$$\forall j_i, \quad j_i \; is \text{ non-preemptive.} \tag{4}$$

Notice that the constraint in (2) is for the available capacity constraint, i.e., actually allocated MapReduce slots to any job is not greater than the number of available MapReduce slots in the system, (3) is the non-overlapping time constraint of Map and Reduce stage for a single job, i.e., for the same job, the start time of its reduce stage should not earlier than the end-time of its map stage (this is reasonable because we consider that reduce stage includes combing, sorting, shuffle and outputing final results), and (4) considers the non-preemptive tasks. Based on the problem formulation, we propose a new approach to minimize the makespan of a set of given MapReduce jobs.

3 HScheduler: a new approach to minimize the makespan in MapReduce

3.1 Offline scheduling

The original Johnson's algorithm [13] considers that "There are n items which must go through one production stage or machine and then a second one. There is only one machine for each stage. At most one item can be on a machine at a given time". One challenging issue is that we cannot directly apply Johnson algorithm to MapReduce. To adapt the MapReduce model, we treat the Map and Reduce slots (stage resources) respectively as a whole (like a single machine) and consider Map and Reduce as two-stage non-overlapped for each job, then we can apply Johnson's algorithm to find the lower bound. Let us consider a collection of n jobs, where each job j_i is represented by the pair (m_i, r_i) of map and reduce stage durations, respectively. Each job $j_i = (m_i, r_i)$ with an attribute S_i defined as follows:

$$S_i = \begin{cases} (m_i, m), & \text{if } \min(m_i, r_i) = m_i, \\ (r_i, r), & \text{otherwise} \end{cases} \tag{5}$$

The first argument in S_i (i.e., m_i or r_i) is called the stage duration and denoted as S_i^1. The second argument (i.e., m or r) is called the stage type (map or reduce) and denoted as S_i^2. Notice that, when all $r_i = 0$, Johnson's algorithm reduces to the shortest process time first (SPT) algorithm, which is known to be optimal for minimizing total finish time of all jobs.

Algorithm 3.1 presents the pesudo-code of revised Johnson's algorithm to find the lower bound of the makespan in multiple MapReduce jobs. It first sorts all the n jobs from the original set J in the ordered list L in such a way that job j_i precedes job j_{i+1} if and only if $\min(m_i, r_{i+1}) \leq min(m_{i+1}, r_i)$, the proof is provided in Lemma 2. It finds the smallest value among all durations, if the stage type in S_i is m, i.e., it represents the Map stage, then the job j_i is placed at the head of the schedule; otherwise, j_i is placed at the tail. Then, the allocated job is removed and other jobs are considered in the same fashion. The complexity of Johnson's algorithm is dominated by the sorting operation and thus is $O(nlogn)$.

Theorem 1 *Johnson's algorithm obtains theoretical lower bound of the makespan of two-stage production system when all jobs go through the same two stages and each job utilizes all resource of each stage. The detailed proof for Theorem 1 is provided in* [13]. *For completeness, we sketch the key points in the following. Let X_i be the idle period of time for the Reduce phase immediately before the ith item comes onto the Reduce phase. If, for example, we consider the job sequence ϕ = 1, 2, 3, ..., n, we have the following time relationship as shown in Fig. 2:*

input : Estimating all Jobs' Map and Reduce durations (m_i, r_i) [1] by utilizing all available Map and Reduce slots in the system, the total number MapReduce slots (P_m^{max}, P_r^{max}) for a Hadoop cluster
output: scheduled jobs, makespan
List the Map and Reduce's durations in two vertical columns (implemented in a list);
for *all* $j_i \in J$ **do**
  Find the shortest one among all durations (min (m_i, r_i));
  In case of ties, for the sake of simplicity, order the item with the smallest subscript first. In case of a tie between Map and Reduce, order the item according to the Map;
  IF it is first Map type, place the corresponding item at the first place;
  ELSE it is first Reduce type, place the corresponding item at the last place;
  IF it is Map type, place the corresponding item right next to the previous job (i.e., in non-decreasing order);
  ELSE it is Reduce type, place the corresponding item left next to the previous job (i.e., in non-increasing order);
  Remove both time durations for that task;
  Repeat these steps on the remaining set of items;
end
Compute the makespan (C_{max});

Algorithm 3.1: Revised Johnson algorithm (RJA Algorithm) for the lower bound

$$
\begin{aligned}
X_1 &= m_1 \\
X_2 &= \max(m_1 + m_2 - r_1 - X_1, 0), \\
X_1 + X_2 &= \max(m_1 + m_2 - r_1, m_1) \\
X_3 &= max\left(\sum_{i=1}^{3} m_i - \sum_{i=1}^{2} r_i - \sum_{i=1}^{2} X_i, 0\right) \\
\sum_{i=1}^{3} X_3 &= \max\left(\sum_{i=1}^{3} m_i - \sum_{i=1}^{2} r_i, \sum_{i=1}^{2} X_i\right) \\
&= \max\left(\sum_{i=1}^{3} m_i - \sum_{i=1}^{2} r_i, \sum_{i=1}^{2} m_i - r_1, m_1\right)
\end{aligned}
\tag{6}
$$

In general,

$$
\sum_{i=1}^{n} X_i = \max_{j=1}^{n} K_j \tag{7}
$$

where

$$
K_j = \sum_{i=1}^{j} m_i - \sum_{i=1}^{j-1} r_i. \tag{8}
$$

Let

$$
F(\phi) = \max_{j=1}^{n} K_j \tag{9}
$$

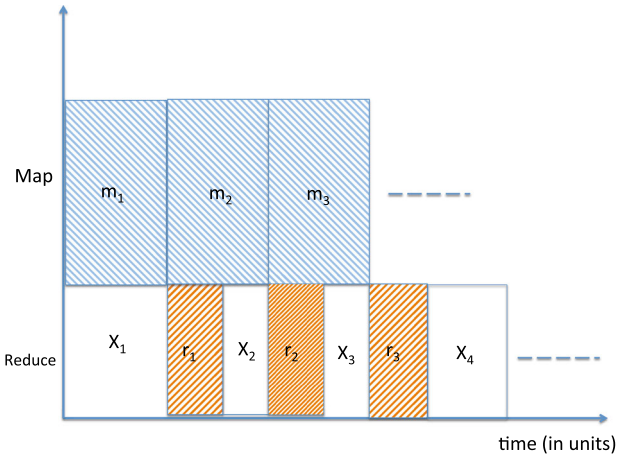

Fig. 2 Two-stage diagram for MapReduce jobs

where ϕ is the job sequence. To minimize F, we need a job sequence ϕ^* such that $F(\phi^*) \leq F(\phi_0)$ for any ϕ_0.

Lemma 1 *An optimal ordering is given by the following rule (I), job j precedes job* $j+1$ *if*

$$\min(m_i, r_{i+1}) < \min(m_{i+1}, r_i) \tag{10}$$

Lemma 1 is proved in [13].

Lemma 2 *Rule (I) is transitive.*

Let consider there are three jobs 1, 2, and 3. Suppose min $(m_1, r_2) \leq$ min (m_2, r_1) and min $(m_2, r_3) \leq$ min (m_3, r_2). Then, min $(m_1, r_3) \leq$ min (m_3, r_1) except possibly when job 2 is indifferent to both job 1 and 3.

Proof Case 1: $m_1 \leq r_2, m_2, r_1$ *and* $m_2 \leq r_3, m_3, r_2$; then $m_1 \leq m_2 \leq m_3$ and $m_1 \leq r_1$ so $m_1 \leq \min(m_3, r_1)$.
Case 2: $r_2 \leq m_1, m_2, r_1$ and $r_3 \leq m_2, m_3, r_2$; then $r_3 \leq r_2 \leq r_1$ and $r_3 \leq m_3$ so $r_3 \leq \min(m_3, r_1)$.
Case 3: $m_1 \leq r_2, m_2, r_1$ and $r_3 \leq m_2, m_3, r_2$; then $m_1 \leq r_1$ and $r_3 \leq m_3$ so that $\min(m_1, r_3) \leq \min(m_3, r_1)$.
Case 4: $r_2 \leq m_1, m_2, r_1$ and $m_2 \leq r_3, m_3, r_2$; then $m_2 = r_2$ and we have job 2 indifferent to job 1 and job 3. In this case, job 1 may or may not precede job 3 but there is no contradiction to transitivity as long as we order job 1 and job 3 first, then put job 2 anywhere. □

The rule (I) is transitive (also proved in [13]), thus leading to a job sequence ϕ^*, which is unique. Then, $F(\phi^*) \leq F(\phi_0)$ for any job sequence ϕ_0 since ϕ^* can be obtained from phi_0 by successive interchanges of consecutive jobs according rule (I), and each interchange will give a value of F smaller than or the same as before (Fig. 3).

Fig. 3 Five MapReduce Jobs Examples

	m_i	r_i	S_i
J_1	4	5	(4, m)
J_2	1	4	(1, m)
J_3	6	30	(6, m)
J_4	30	4	(4, r)
J_5	2	3	(2, m)

(a)

	m_i	r_i	S_i
J_2	1	4	(1, m)
J_5	2	3	(2, m)
J_1	4	5	(4, m)
J_3	6	30	(6, m)
J_4	30	4	(4, r)

(b)

From Revised Johnson's algorithm, we can obtain theoretical lower bound of the makespan as follows:

(1) Considering there are n tasks for map and reduce stages. Let m_i be work time of the ith task on Map phase, and r_i be the corresponding time on the Reduce phase for a given Hadoop cluster with $P_m^{\max} \times P_r^{\max}$ MapReduce slots.
(2) Then, the optimal total elapsed time (makespan) is

$$T = \sum_{i=1}^{n} r_i + \max_{u=1}^{n} K_u \tag{11}$$

where

$$K_u = \sum_{i=1}^{u} m_i - \sum_{i=1}^{u-1} r_i. \tag{12}$$

Observation 1 *If each job utilizes all map and all reduce slots during its processing, there is a match between the assumptions of the classic Johnson algorithm for two-stage production system and MapReduce jobs, then RJA Algorithm can be applied to find the theoretical lower bound of minimizing the makespan of all MapReduce jobs.*

Example 1 For the five MapReduce jobs given in [1], we reproduce it in Fig. 2a, b, where (a) shows the durations of Map and Reduce stages of each job, and (b) provides ordered list of the five jobs by applying Johnson's algorithm. All jobs request 30 × 30 MapReduce slot. According to Johnson's algorithm, the optimal sequence is $\phi = (2, 5, 1, 3, 4)$. The total delay time for this sequence can be computed using Eq. 12, i.e., 1 units, and the total elapsed time (makespan) is 47 units (using Eqs. (11, 12)). If one reverses the order of the jobs, then the worst-case result can be obtained (as the upper bound), i.e., 78 units.

Observation 2 *The job duration at each stage closely depends on the amount of allocated Map and Reduce slots. If the system scheduler allocates more or less number of MapReduce slots than the required slots, a jobs' duration can be changed.*

Observation 2 is straightforward practice from MapReduce framework.

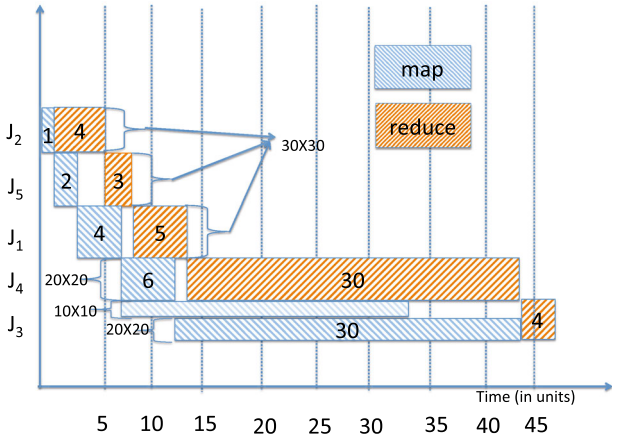

Fig. 4 Five MapReduce Jobs Execution in One Cluster

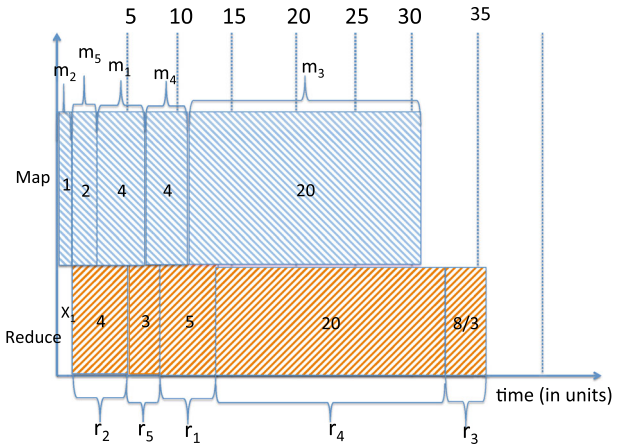

Fig. 5 New Result of Five MapReduce Jobs Execution

Example 2 Consider Example 1 again, now let jobs j_1, j_2, and j_5 be comprised of 30 map and 30 reduce tasks, and jobs j_3 and j_4 consist of 20 map and 20 reduce tasks, other parameters are the same as in Example 1. We reproduce results in Fig. 4 that visualizes the execution of these five MapReduce jobs according to the generated Johnson's schedule, $\phi = (j_2, j_5, j_1, j_3, j_4)$. For job j_3 and j_4, we allow that any job can use all available MapReduce slots in the system when execution (*this can be implemented easily in Hadoop, for example by splitting the large input files based on available number of MapReduce slots. The configuration APIs include mapred.task.tracker.map.tasks.maximum, mapred.task.tracker.reduce.tasks.maximum*

and conf.setNumReduceTasks. Also, changing chunk size of the input file is possible before uploading to HDFS). Now job j_3 and j_4 can use all available 30×30 MapReduce slots, then j_3 will have Map and Reduce durations $(20, \frac{8}{3})$, respectively; j_4 will have Map and Reduce durations (4, 20), respectively, both shorter than only using 20×20 MapReduce slots. Therefore, the makespan will be $35\frac{2}{3}$ as shown in Fig. 5, where $X_1 = 1$. This result is about 12 % smaller than the result (40 time units) obtained by two-pool approach [1], and is about 31.7 % smaller than the result (47 time units) where total available slots are not fully utilized. Therefore, the following principle is the key strategy for our results.

Claim 1 *The system scheduler can decrease or increase the allocated number of MapReduce slots to a job based on the total available MapReduce slots in the system.*

Using Hadoop APIs [2] explained in Example 2, the scheduler can easily adjust the MapReduce slots. Assuming that there are $P_m^{\max} \times P_r^{\max}$ MapReduce slots in the given Hadoop cluster, there are two Jobs A and B, each has requested MapReduce slots (R_m^A, R_r^A) and (R_m^B, R_r^B), respectively. Their allocated MapReduce slots can be adjusted based on $P_m^{\max} \times P_r^{\max}$. Notice that their theoretical makespan $C_{\max}$ can be easily computed using Eqs. (11, 12) directly. In the following, we consider offline and online scheduling, respectively.

3.2 Offline HScheduler

Based on Theorem 1, Observation 1 and Claim 1, we design a new approach called HScheduler for efficiently scheduling of MapReduce jobs to minimize makespan. Algorithm 3.2 presents the pesudo-code of Offline HScheduler algorithm. It first allocates all available MapReduce slots to a given set of jobs by recomputing their actual durations based on available slots. This is to change their Map and Reduce durations by taking more or less execution waves based on available slots. Then, it calls RJA algorithm to schedule all updated jobs.

Theorem 2 *HScheduler is optimal regarding minimizing the makespan of a given set of jobs when considering Map and Reduce phases only.*

Proof Since HScheduler satisfies the condition of Johnson's algorithm, and from Theorem 1 we know that Johnson's algorithm is optimal for two-stage MapReduce-like systems when Observation 1 is satisfied; therefore, HScheduler is optimal given a set of jobs when considering Map and Reduce phases.

This completes the proof. □

The complexity of HScheduler is dominated by Johnson's algorithm and thus is $O(nlogn)$.

input : The total number of slots $P_m^{max} \times P_r^{max}$ for a Hadoop cluster, all Jobs' Map and Reduce durations estimated by the methods suggested in [1] through utilizing all available MapReduce slots
output: scheduled jobs, makespan
1 **for** *all jobs* **do**
2 IF a job's required slots $R_m^i \geq P_m^{max}$, or $R_r^i \geq P_r^{max}$ (total available slots), THEN allocates all available slots to it and adds more execution waves (waves=max($\lceil R_m^i / P_m^{max} \rceil$,$\lceil R_r^i / P_r^{max} \rceil$) based on tasks' splitting of large input files on Map file block size;
3 ELSE ajust MapReduce slot settings and allocates all available slots to it and records actual execution waves;
4 End;
5 **end**
6 Call RJA Algorithm;
7 Compute the makespan (C_{max});

Algorithm 3.2: Offline HScheduler

3.3 Online HScheduler

To evaluate the performance of online algorithm, a well-known approach is to compare its results against theoretical (optimal) results obtained in offline scheduling.

Definition 4 The competitive ratio is the ratio of the makespan of mutilple online MapReduce jobs over the optimum makespan of these jobs considered in offline scheduling in which all durations are known a priori and their executing order can be arranged in advance.

Algorithm 3.3 presents the pesudo-code of OnHScheduler algorithm for online MapReduce. All online jobs are proceeded in First-in-First-out (FIFO) manner. It first allocates all available MapReduce slots to online jobs by recomputing their actual durations based on available slots. This is to change their Map and Reduce durations by taking more or less execution waves based on available slots. *Notice that this is different from FIFO which does not adjust MapReduce slots based on available slots*. Then according to the arrival time of the jobs, OnHScheduler allocates MapReduce slots to jobs.

input : The total number of slots $P_m^{max} \times P_r^{max}$ for a Hadoop cluster, a Jobs' Map and Reduce durations estimated by the methods suggested in [1] through utilizing all available MapReduce slots
output: scheduled jobs, makespan
1 **for** *all jobs* **do**
2 IF a job's required slots $R_m^i \geq P_m^{max}$, or $R_r^i \geq P_r^{max}$ (total available slots), THEN allocates all available slots to it and adds more execution waves (waves=max($\lceil R_m^i / P_m^{max} \rceil$,$\lceil R_r^i / P_r^{max} \rceil$) based on tasks' splitting of large input files on Map file block size;
3 ELSE ajust MapReduce slot settings and allocates all available slots to it and records actual execution waves;
4 End;
5 **end**
6 Compute the makespan (C_{max}).

Algorithm 3.3: OnHScheduler

Next, we analyze the competitive ratio of OnHScheduler.

Theorem 3 *OnHScheduler is 2-competitive.*

Proof The two-stage diagram of MapReduce is shown in Fig. 2, where X_i is the delay time between two consecutive Reduce jobs. Using Eq. (11) and the feature of OnHScheduler, the proof is as follows.

Set $K_{\text{on}} = \max_{u=1}^{n} K_u^{\text{on}}$ and $K_{\text{opt}} = \max_{u=1}^{n} K_u^{\text{opt}}$ respectively for OHLScheduler and optimal result in Eq. (11), which is the sum of delay time in the second (Reduce) phase. Set the makespan of OnHScheduler and optimal result as T_{on} and T_{opt} respectively, then

(i) Assuming that $\sum_{i=1}^{n} m_i \geq \sum_{i=1}^{n} r_i$, from Eqs. (11, 12), we know that $T_{\text{opt}} = \sum_{i=1}^{n} r_i + K_{\text{opt}} = \sum_{i=1}^{n} m_i + \max_{i=1}^{n} r_i$. K_{opt} will be very small (or close to zero) for optimal result and K_{on} in the worst case will be close to $\sum_{i=1}^{n} m_i$, so

$$\begin{aligned} \frac{T_{\text{on}}}{T_{\text{opt}}} &= \frac{\sum_{i=1}^{n} r_i + K_{\text{on}}}{\sum_{i=1}^{n} r_i + K_{\text{opt}}} \\ &\leq \frac{\sum_{i=1}^{n} r_i + \sum_{i=1}^{n} m_i}{\sum_{i=1}^{n} m_i + \max_{i=1}^{n} r_i} \\ &\leq 1 + \frac{\sum_{i=1}^{n-1} r_i}{\sum_{i=1}^{n} m_i + \max_{i=1}^{n} r_i} \\ &\leq 2 \left(\text{since} \sum_{i=1}^{n} m_i \geq \sum_{i=1}^{n} r_i \right) \end{aligned} \tag{13}$$

(ii) Assuming that $\sum_{i=1}^{n} m_i < \sum_{i=1}^{n} r_i$, we know $K_{\text{on}} \leq \sum_{i=1}^{n} r_i$, then

$$\begin{aligned} \frac{T_{\text{on}}}{T_{\text{opt}}} &= \frac{\sum_{i=1}^{n} r_i + K_{\text{on}}}{\sum_{i=1}^{n} r_i + K_{\text{opt}}} \\ &\leq \frac{\sum_{i=1}^{n} r_i + \sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} r_i + K_{\text{opt}}} \\ &\leq 2 \left(\text{since} 0 < K_{\text{opt}} \leq \sum_{i=1}^{n} r_i \right) \end{aligned} \tag{14}$$

This completes the proof. □

Theorem 4 *The upper bound of 2-competitive is tight for OnHScheduler.*

Proof Considering the following worst case for OnHScheduler. There are $2n$ jobs, the first n jobs are Map types with Map duration $m_i = C_{\max}$ and Reduce duration $r_i = nt$; and other n jobs are Reduce types with Map duration $m_i = nt$ and Reduce duration $r_i = C_{max}$. By applying Johnson's algorithm, the optimal makespan is $T_{\text{opt}} = (1+n+n^2)C_{\max}$ using Eqs. (11, 12) and Algorithm 3.1. On the other hand, supposing

Fig. 6 The example of online jobs

J_i	m_i(map duration)	r_i(reduce duration)	t_i(arrived time)
J_1	4	5	0
J_2	1	4	1
J_3	30	4	2
J_4	6	30	3
J_5	2	3	4

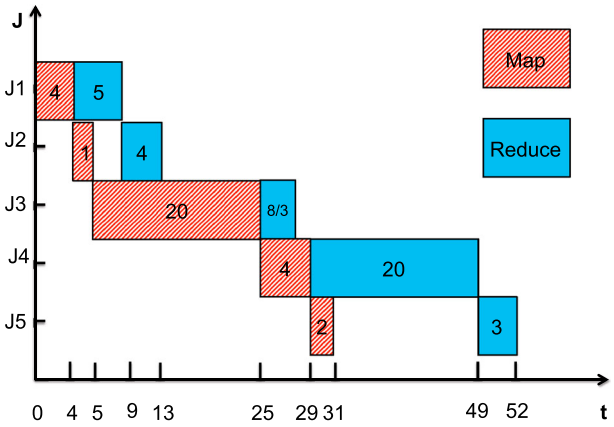

Fig. 7 The scheduling result of online jobs

all jobs come in reverse order (the worst case for OnHScheduler), we can easily find that the makespan is $T_{\text{on}} = (1 + 2n^2)C_{\max}$. So the competitive ratio in this case is

$$\frac{T_{\text{on}}}{T_{\text{opt}}} = \frac{(1 + 2n^2)t}{(1 + n + n^2)t} \approx 2, \quad \text{as} \quad n \to \infty. \tag{15}$$

□

Example 3 For the five MapReduce jobs in Example 1, we reproduce it in Fig. 6 where the arrival time of each job is denoted as t_i for online scheduling. Assuming that 30×30 MapReduce slots are used for all jobs. Figure 7 provides scheduling results by OnHScheduler for these online jobs with a total makespan of 52 units, which is 5 units larger than the optimal offline result of 47 units.

4 Performance evaluation

In this section, we conduct performance evaluation based on real data traces of Word-counts [14] and Terasort [12]. Note that it is possible to estimate their durations for

different types of MapReduce jobs in a given configuration of Hadoop cluster, a solution for the estimation is provided in [1].

4.1 Configuration of Hadoop cluster

The Hadoop cluster (YARN version) in the test is formed by four Dell R720 servers connected by 1 Gbs Eithernet LAN. One Master node and 32 Slave nodes are set. There are 8 virtual machine (VM) data nodes on each of four servers. Each server has 2 Intel Xeon 2520, 32 GB memory, 1TB hard disk and each VM node has 2 GB memory and 160 GB hard disk. Altogether 64 $\times$ 64 MapReduce slots are created similar to the environment introduced in [1] so that comparative study can be conducted.

4.2 Replay with real data traces

We use the similar workloads as in [1] in our experiments so that comparative study can be conducted:

- This workload represents a mixed number of MapReduce jobs that is based on the analysis performed on the Yahoo! M45 cluster [1]. The number of Map and Reduce tasks is generated by Normal distribution; the durations of Map and Reduce phases are obtained from real data of Wordcount [14] and TeraSort [12].
- Unimodel: where a set of 50 Wordcount [14] (with mean Map duration of 65 s and mean Reduce duration of 57 s uniformly distributed) and 50 Terasot jobs [12] (with mean Map duration of 73 s and Reduce duration of 58 s uniformly distributed) are tested, it uses a single scale factor for the overall workload, i.e., the scale factor for each job is drawn uniformly from [1,9], and normal distribution with parameter ($N(154, 558) \times 0.1$) for the number of Map tasks and ($N(19, 145) \times 0.1$) for Reduce tasks.
- Bimodel: where a subset of 20 Wordcounts from [14] (with mean Map duration of 448 s and mean Reduce duration of 413 s uniformly distributed) and 20 Terasot jobs from [12] (with mean Map duration of 287 s and Reduce duration of 306 s uniformly distributed). In this case, 80 % of jobs are scaled using a factor uniformly distributed between [1,10] and the remaining jobs (20 %) are scaled using [4,9], and Normal distribution with parameter round ($N(154, 558) \times 0.3$) for the number of Map tasks and round ($N(19, 145) \times 0.3$) for Reduce tasks. This mimics workloads that have a large fraction of short jobs and a small subset of long jobs.

All results are obtained by the average of 10 runs.

4.3 Different scheduling algorithms compared

We compare the following algorithms:

- *Random Order (Rand)*: this algorithm just schedules all jobs in a random order of their job IDs.

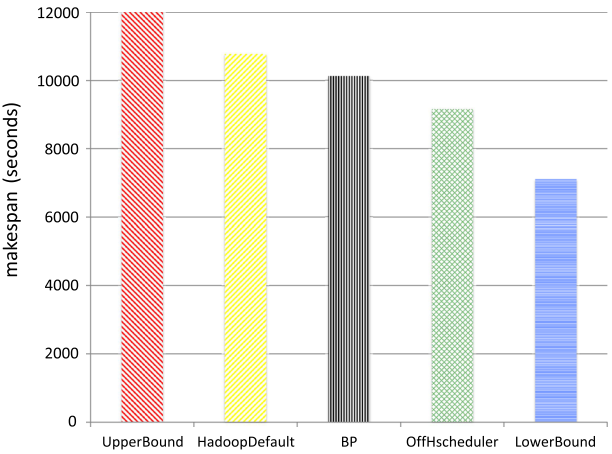

Fig. 8 The comparison of offline makespn in unimodel (seconds)

- *The Lower Bound (LowerBound)*: this is computed by revised Johnson's algorithm, which works as a theoretical lower bound because it does not consider additional process time caused by jobs' setting up, dispatch and migration in the real Hadoop cluster.
- *The Upper Bound (UpperBound)*: this algorithm schedules all jobs in reversed order of revised Johnson algorithm. This provides the worst case (upper bound) regarding makespan of all jobs and is proved in [13].
- *BalancedPools (BP)* is another way to minimize the makespan for offline scheduling proposed in [1], it partitions the Hadoop cluster into two balanced pools and then allocated each job to a suitable pool to minimize the makespan.
- *HScheduler*: our proposed algorithms, HScheduler for offline and OnHScheduler for online, respectively.

In all tests, 32 data nodes each with 2 MapReduce slots are set, 2 pools each with 22 and 42 MapReduce slots are set respectively for BalancedPools (BP) algorithm.

For offline scheduling, Figs. 8 and 9 present the makespan comparison of four algorithms; the UpperBound (Reversed Order Jonhson's algorithm) is the worst case, working as the upper bound of makespan while results obtained from Johnson's algorithm are the theoretical lower bounds (LowerBound). Random-order scheduling and UpperBound have higher makespan than HScheduler. HScheduler has 8–10 % less makespan on average than BP(BalancedPools). HScheduler is on average 15 and 13 % larger than theoretical lower bounds in Unimodel and Bimodel, respectively. This is because HScheduler has additional process time such as job setting up, dispatch and migration, and all map and reduce jobs may not finish at the same time in real Hadoop environment. Observation 2 is also validated. In Fig. 10, we also conduct tests by 50 Terasort data (with mean Map duration of 73 s and Reduce duration of 58 s uniformly distributed) without considering Bimodel or Unimodel. From extensive real experiments, similar results are observed.

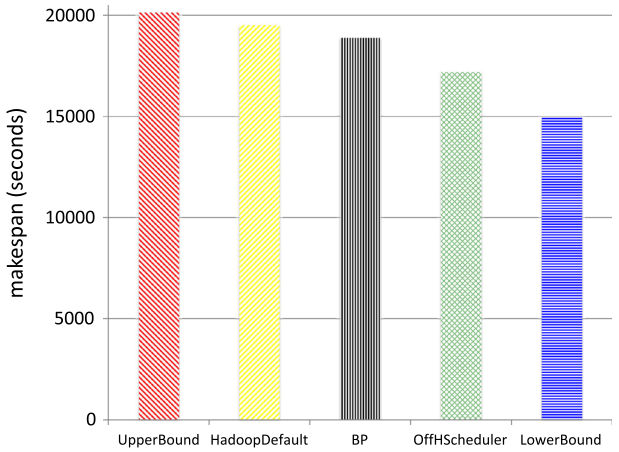

Fig. 9 The comparison of offline makespn in bimodel (seconds)

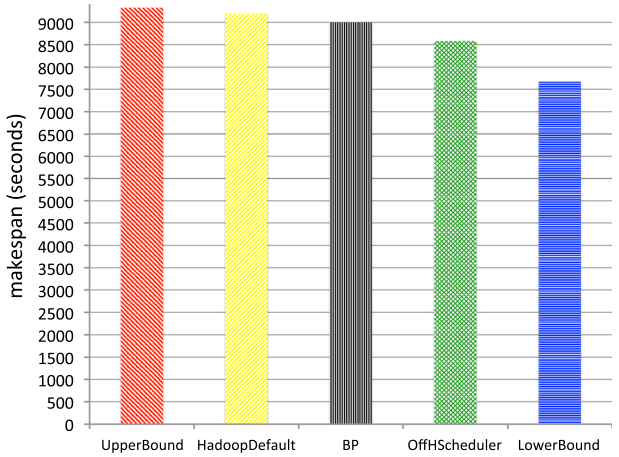

Fig. 10 The comparison of offline makespan (seconds) of 50 Terasorts Jobs

As for online scheduling, Figs. 11 and 12 present the makespan comparison of four algorithms; the UpperBound (Reversed Order Jonhson's algorithm) is the worst case, working as the upper bound of makespan while results obtained from Johnson's algorithm are the theoretical lower bounds. Random-order scheduling and UpperBound have higher makespan than OnHScheduler. OnHScheduler has 10–11 % less makespan on average than Rand. OnHScheduler is on average 10 and 15 % larger than Johnson lower bound in Unimodel and Bimodel, respectively. This is because OnHScheduler has additional process time such as job setting up, dispatch and migration, and the waiting time in real Hadoop environment.

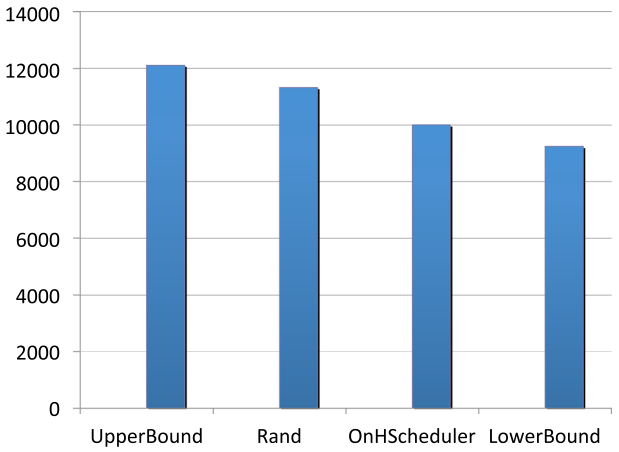

Fig. 11 The comparison of online makespan in unimodel (seconds)

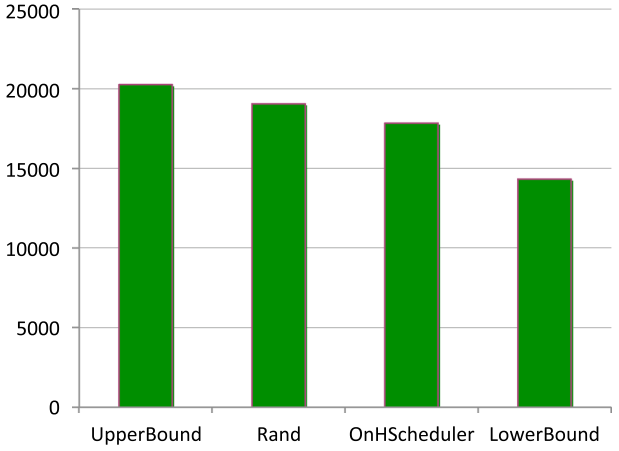

Fig. 12 The comparison of online makespan in bimodel (seconds)

5 Conclusions and further work

In this work, by adopting a new strategy, implementation of allocating available MapReduce slots, and combining the features of classical Johnson's algorithm, we propose and validate new scheduling algorithms for MapReduce framework to minimize the makespan. The proposed approaches have better performance in comparison to other algorithms in the literature. The HScheduler can be applied to improve energy efficiency, throughput, and response time in data centers. There are a few research directions under consideration:

– Providing evaluation by classifying different types of MapReduce jobs. For example, we can consider three general types: compute-intensive (such as sorting), memory-intensive (such as wordcounts) and IO-intensive (such as searching). For

each type of jobs, we schedule them to a separate Hadoop cluster to compare the makespan and other performance metrics.
– Considering more performance metrics than the makespan. Energy efficiency metrics such as total power and energy consumptions can be added; Load-balance metrics such as average utilization and skewness can be considered too.
– Proposing approaches to consider priorities such as preemption. In this paper, non-preemptive case is discussed and we are extending our study to preemptable job scenario and other priority strategies.

Acknowledgments This research is partially supported by China National Science Foundation (CNSF) with project ID 61450110440 and Sichuan Province Technology Plan (ID 2016GZ0322); Chongqing Research Program of Basic Research and Frontier Technology (ID cstc2015jcyjB0244). Prof. Wenhong Tian finished most of this work when he was a visiting fellow at CLOUDS lab led by Prof. Rajkumar Buyya at the University of Melbourne, Australia. The author thanks team members in CLOUDS for their comments to polish the manuscript.

References

1. Verma A, Cherkasova L, Campbell RH (2013) Orchestrating an ensemble of MapReduce jobs for minimizing their makespan. IEEE Trans Depend Secure Comput (online version)
2. Capacity Scheduler Guide. Available http://hadoop.apache.org/common/docs/r0.20.1/capacity~scheduler.html
3. 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 schedul- ing. In: Proceeding of EuroSystem, ACM, pp 265–278
4. Wolf J et al (2010) FLEX: a slot allocation scheduling optimizer for MapReduce Workloads. In: ACM/IFIP/USENIX international middleware conference, Lecture Notes in Computer Science, vol 6452, pp 1–20
5. Verma A, Cherkasova L, Campbell RH (2011) ARIA: automatic resource inference and allocation for mapreduce environments. In: Proceedings of the ICAC, Germany, pp 235–244
6. Verma A, Cherkasova L, Campbell RH (2011) Play it again, SimMR! In: Proceedings of the international IEEE Cluster'2011, IEEE Computer Society Washington, DC, USA, pp 253–261
7. Zhu Y, Jiang Y, Wu W, Ding L, Teredesai A, Li D, Lee W (2014) Minimizing makespan and total completion time in MapReduce-like systems. In: Proceedings of INFOCOM, Toronto, ON, pp 2166–2174, 27 April 2014–2 May 2014
8. Herodotou H, Babu S (2011) Profiling, what-if analysis, and cost based optimization of MapReduce programs. In: Proceedings of the VLDB Endowment 4(11):1111–1122
9. Moseley B, Dasgupta A, Kumar R, Sarl T (2011) On scheduling in map-reduce and flow-shops. In: Proceedings of SPAA, ACM New York, NY, pp 289–298
10. Verma A, Cherkasova L, Campbell RH (2012) Two sides of a coin: optimizing the schedule of MapReduce jobs to minimize their makespan and improve cluster performance. In: MASCOTS, IEEE Computer Society, pp 11–18
11. Zheng Y, Shroff NB, Sinha P (2013) A new analytical technique for designing provably efficient MapReduce schedulers. In: The Proceedings of INFOCOM, Turin, pp 1600–1608, 14–19 April 2013
12. http://sortbenchmark.org/YahooHadoop.pdf
13. Johnson S (1954) Optimal two-and three-stage production schedules with setup times included. Naval Res Log Q
14. Wordcount http://www.cs.cornell.edu/home/llee/data/simple/
15. Garey M, Johnson D (1979) Computers and intractability: a guide to the theory of NP-completeness. WH Freeman & Co, New York