Gandomi *et al. J Big Data* (2019) 6:106
https://doi.org/10.1186/s40537-019-0253-9

Journal of Big Data

RESEARCH Open Access

HybSMRP: a hybrid scheduling algorithm in Hadoop MapReduce framework

Abolfazl Gandomi[1], Midia Reshadi[1*], Ali Movaghar[2] and Ahmad Khademzadeh[3]

*Correspondence:
reshadi@srbiau.ac.ir
[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
Full list of author information is available at the end of the article

Abstract

Due to the advent of new technologies, devices, and communication tools such as social networking sites, the amount of data produced by mankind is growing rapidly every year. Big data is a collection of large datasets that cannot be processed using traditional computing techniques. MapReduce has been introduced to solve large-data computational problems. It is specifically designed to run on commodity hardware, and it depends on dividing and conquering principles. Nowadays, the focus of researchers has shifted towards Hadoop MapReduce. One of the most outstanding characteristics of MapReduce is data locality-aware scheduling. Data locality-aware scheduler is a further efficient solution to optimize one or a set of performance metrics such as data locality, energy consumption and job completion time. Similar to all situations, time and scheduling are the most important aspects of the MapReduce framework. Therefore, many scheduling algorithms have been proposed in the past decades. The main ideas of these algorithms are increasing data locality rate and decreasing the response and completion time. In this paper, a new hybrid scheduling algorithm has been proposed, which uses dynamic priority and localization ID techniques and focuses on increasing data locality rate and decreasing completion time. The proposed algorithm was evaluated and compared with Hadoop default schedulers (FIFO, Fair), by running concurrent workloads consisting of Wordcount and Terasort benchmarks. The experimental results show that the proposed algorithm is faster than FIFO and Fair scheduling, achieves higher data locality rate and avoids wasting resources.

Keywords: MapReduce, Scheduling, Hybrid algorithm, Data Locality, Dynamic priority

Introduction

Distributed and parallel processing is one of the best intelligent ways to store and compute big data [1]. Most definitions defined big data as characterized by the 3Vs: the extreme volume of data, the wide variety of data types and the velocity at which the data must be processed. MapReduce [2] is a programming model for big data processing. MapReduce programs are intrinsically parallel [3, 4]. MapReduce executes the programs in two phases, map and reduce, so that each phase is defined by a function called mapper and reducer. A MapReduce framework consists of a master and multiple slaves. The master is responsible for the management of the framework, including user interaction, job queue organization and task scheduling. Each slave has a fixed number of map and reduce slots to perform tasks. The job scheduler located in the master assigns tasks according to the number of free task slots

reported by each slave through a heartbeat signal [5]. Each job is split into a large number of map and reduce tasks before being started. The runtime is in charge of running tasks for every job until they are completed. The tasks are actually executed in any of the slave nodes which comprise the MapReduce cluster. In particular, the task scheduler is responsible for deciding what tasks are run at each moment in time, as well as what slave node will host the task execution [6]. A number of scheduling algorithms are available which help Hadoop [7] to improve its performance in different factors such as Data locality rate and job completion time. Each of the traditional algorithms improves the performance regarding certain factors. In Hadoop, data locality is the process of moving the computation close to where the actual data resides on the node, instead of moving large data to computation. This minimizes network congestion and increases the overall throughput of the system.

Hadoop provides three built-in scheduling modules [7]: First In First Out (FIFO) scheduler, Fair scheduler, and Capacity scheduler. These schedulers have different performances such as execution time and waiting time in different situations. Limitations of Hadoop FIFO occur when short jobs have to wait too long behind long running jobs, thus negatively affecting the job response time. The Hadoop Fair scheduler, developed by Zaharia et al. [8], was the first to addresses this limitation in depth through a fair share mechanism between multiple concurrent jobs. Over time, fair scheduler assigns resources such that all jobs get, on average, an equal share of resources. Additionally, fair scheduler extends the data locality of FIFO using a delayed execution mechanism. The most negative point of this algorithm is that it does not consider the job priority of each node. There are many Hadoop MapReduce scheduling algorithms which focus on MapReduce scheduling issues, some of which use only one technique to achieve high data locality rate while some others use dynamic job priority to decrease response and completion time. For example, matchmaking algorithm [9] uses a locality ID to increase the data locality rate, while HybS [10] uses a dynamic job priority. Due to the fact that many of the existing MapReduce scheduling algorithms do not use these techniques together, the current paper has proposed a new Hybrid Scheduling Algorithm, called HybSMRP (Hybrid Scheduling MapReduce priority), which uses a data localization technique and Dynamic job priority to increase data locality rate and decrease completion time.

The rest of this paper is organized as follows. In the section on research Background, an overview of Hadoop MapReduce is described. The section entitled Related Works provides a review of the related works. It describes existing approaches and introduces the background of Hadoop MapReduce scheduling algorithms. The section on Proposed Method presents the scheduling algorithm introduced in this work. Evaluation setting and results are given in the Results and Discussion Section. Finally, the section on Conclusion and Future Work concludes the paper with remarks on the limitations and possible future works.

Background

In this section, firstly an overview of Hadoop MapReduce is presented and, then, MapReduce scheduling issues and Hadoop default scheduling algorithms are introduced.

Overview of Hadoop and MapReduce

MapReduce is the processing layer of Hadoop. MapReduce programming model is designed for processing large volumes of data in parallel by dividing the job into a set of independent tasks. In Hadoop MapReduce, user defines the two functions, map and reduce, and generally uses Hadoop Distributed File System (HDFS); consequently, I/O performance of a job can depend on HDFS. Hadoop MapReduce works on a set of key-value pairs; then, the inputs enter in form of key-value pairs. In the first phase, the list of pairs is broken down and the mapper function is called for each of these pairs. Then, Hadoop MapReduce partitions the pairs generated by map phase through the key *k*. Map tasks write their output to the local disk, not to HDFS. In the next phase, the reducer function is invoked for every key k and corresponding values list from the map output [7]. Hadoop uses HDFS for data storing and MapReduce for processing the data. HDFS is Hadoop's implementation of a distributed file system. It is designed to hold a large amount of data and provides access to this data to many clients distributed across a network. HDFS consists of multiple DataNodes for storing data and a master node called NameNode for monitoring DataNodes and maintaining all the meta-data. MapReduce is a programming model and an associated implementation for processing large datasets. It enables users to specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs and a reduce function that merges all the intermediate values associated with the same intermediate key [11].

According to Fig. 1, the map phase is carried out by mappers. Mappers run on unsorted input key/value pairs. Each mapper emits zero, one, or multiple output key/value pairs for each input key/value pairs. The shuffle and sort phase is done by the framework. Data from all mappers are grouped by the key, split among reducers and sorted by the key. Each reducer obtains all values associated with the same key. The programmer may supply custom compare functions for sorting and a partitioner for data split. The partitioner decides which reducer will get a particular key/value pair. The reducer obtains sorted key/value pairs, sorted by the key. The value list contains all values with the same key produced by mappers. Each reducer emits zero, one or multiple output key/value pairs for each input key/value pair.

According to Fig. 2, the output produced by Map is not directly written to disk; rather, it is first written to its memory. It takes advantage of buffering writes in memory. Each map task has a circular buffer memory of about 100 MB by default. When the contents of the buffer reach a certain threshold size (*mapreduce.map.sort.spill.percent*, which has the default value 0.80, or 80%), a background thread will start to spill the contents to disk.

Map outputs will continue to be written to the buffer while the spill takes place, but if the buffer fills up during this time, the map will block until the spill is complete. A new spill file is created every time when the buffer memory reaches the spill threshold, so at last, many spill files can be created. Before the completion of the task, all the spill files will be merged into a single partition and sorted output file. The combine phase is done by combiners. The combiner should combine key/value pairs with the same key. Each combiner may run zero, one, or multiple times.

In Hadoop, all scheduling and allocation decisions are made on a task and node slot level for both the map and reduce phases. The Hadoop scheduling model is a Master/

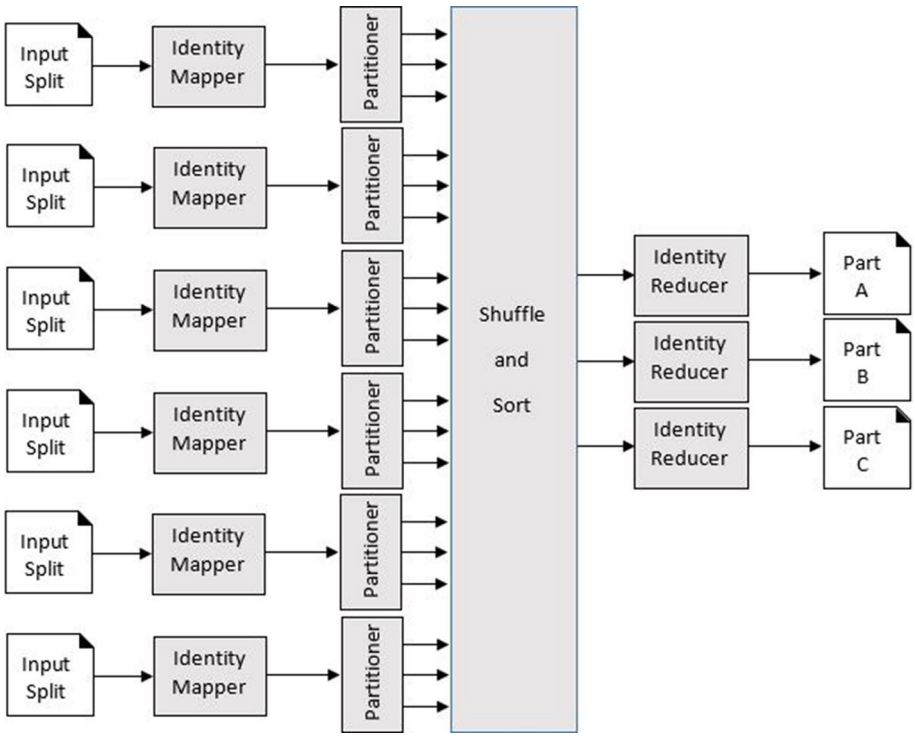

Fig. 1 Processing phases in MapReduce [12]. Mappers run on unsorted input key/value pairs. Each mapper emits zero, one, or multiple output key/value pairs for each input key/value pairs. The shuffle and sort phase is done by the framework. Data from all mappers are grouped by the key, split among reducers and sorted by the key. Each reducer obtains all values associated with the same key

Slave (Master/Worker) cluster structure [14]. The master node (Job Tracker) coordinates the worker machines (Task Tracker). Job Tracker is a process, which manages jobs, and Task Tracker is a process, which manages tasks on the corresponding nodes. The scheduler resides in the Job Tracker and allocates Task Tracker resources to running tasks: map and reduce tasks are granted independent slots on each machine. MapReduce Scheduling system takes on in six steps: firstly, users submit jobs to a queue, and the cluster runs them in order. Secondly, master node distributes Map Tasks and Reduce Tasks to different workers. Thirdly, Map Tasks read the data splits, and run map function on the data which is read in. Next, map Tasks write intermediate results into local disk. Then, reduce Tasks read the intermediate results remotely and run reduce function on the intermediate results which are read in. Finally, these reduce Tasks write the final results into the output files. There are three important scheduling issues in MapReduce, namely locality, synchronization and fairness. Locality is a very crucial issue affecting performance in a shared cluster environment, due to limited network bisection bandwidth. It is defined as the distance between the input data node and task-assigned node. The concept of data locality in MapReduce relates to a time when scheduler tries to assign map tasks to slots available on machines in which the underlying storage layer holds the input intended to be processed. One of the main ideas of the scheduling algorithm proposed here is data localization [15].

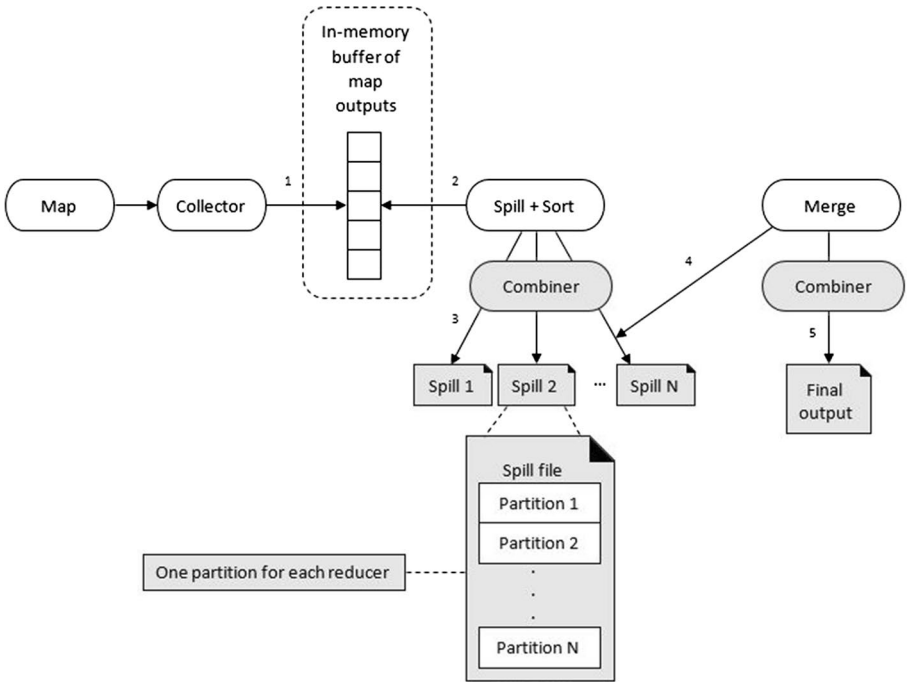

Fig. 2 Spill and Merge phases in MapReduce [13]. The output produced by Map is not directly written to disk, it first writes it to its memory. It takes advantage of buffering writes in memory. Each map task has a circular buffer memory of about 100 MB by default. When the contents of the buffer reach a certain threshold size (mapreduce.map.sort.spill.percent, which has the default value 0.80, or 80%), a background thread will start to spill the contents

Related works

There are many scheduling algorithms which address the main issues of MapReduce scheduling with different techniques and approaches. As it has already been mentioned, some of these algorithms have been focused on improving data locality and some aim to provide synchronization processing. Also, many of these algorithms have been designed to decrease the completion time. LATE [16], SAMR [17], CREST [18], LARTS [19], Maestro [20] and Matchmaking algorithms have focused on data locality. What follows is a brief description of some of the most important algorithms: in Longest Approximate Time to End (LATE) scheduler, backup tasks are used for the tasks that have longer remaining execution time. LATE uses a set of fixed weights to estimate the remaining execution time. This scheduler tries to identify the slow running tasks, and once identified, sends them to another node for execution. If this node is able to perform the task faster, then the system performance increases. The advantage of this method is the calculation of the remaining execution time of the task, together with the calculation of the rate of job progress, which leads to an increase in the rate of system response. In contrast, one of the disadvantages of LATE is that the task selection for re-execution is carried out incorrectly in some cases, which is due to the wrong calculation of the remaining execution time of the task. As a result, Chen et al. [17] recommended Self-Adaptive MapReduce (SAMR) scheduling algorithm, inspired by LATE scheduling algorithm. In this algorithm, the history of job executions is used to calculate the remaining execution time more accurately. The task tracker reads the history

information and adjusts the parameters using this information. The SAMR scheduling algorithm improves MapReduce by saving execution time and system resources. It defined fast nodes and slow nodes to be nodes, which can finish a task in a shorter time and longer time than most other nodes. It will focus on how to account for data locality when launching backup tasks, because data locality may remarkably accelerate the data load and store. Lei et al. [18] proposed a novel approach, CREST (Combination Re-Execution Scheduling Technology), which can achieve the optimal running time for speculative map tasks and decrease the response time of MapReduce jobs. To mitigate the impact of straggler tasks, it is common to run a speculative copy of the straggler task. The main idea is that re-executing a combination of tasks on a group of computing nodes may progress faster than directly speculating the straggler task on target node, due to data locality. Hammoud and Sakr [19] presented another approach discussing the data locality problem. It deals specifically with reduce tasks. The scheduler, named Locality-Aware Reduce Task Scheduler (LARTS), uses a practical strategy that leverages network locations and sizes of partitions to exploit data locality. LARTS attempts to schedule reducers as close as possible to their maximum amount of input data and conservatively switches to a relaxation strategy seeking a balance among scheduling delay, scheduling skew, system utilization, and parallelism. Ibrahim et al. [20] developed a scheduling algorithm called Maestro to avoid the non-local map tasks execution problem that relies on replica aware execution of map tasks. To address this, Maestro keeps track of the chunks and replica locations, along with the number of other chunks hosted by each node. In this way, Maestro was able to schedule map tasks with low impact on other nodes local map tasks execution by calculating the probabilities of executing all the hosted chunks locally. Maestro, with the use of these two waves, provided a higher locality in the execution of map tasks and a more balanced intermediate data distribution for the shuffling phase.

He et al. [9] proposed a matchmaking-scheduling algorithm, which is close to our HybSMRP. Local map tasks are always preferred over non-local map tasks, no matter which job a task belongs to, and a locality marker is used to mark nodes and to ensure each node has a fair chance to grab its local tasks.

Another algorithm, the one closest to our proposed algorithm, is HybS, which was proposed by Nguyen et al. [10]. HybS is based on dynamic priority, in order to reduce the delay for variable length concurrent jobs, and relax the order of jobs to maintain data locality. Also, it provides a user-defined service level value for quality of service. This algorithm is designed for data-intensive workloads and tries to maintain data locality during job execution.

Wang et al. [21] proposed a map task scheduling algorithm in MapReduce with data locality, which is derived an outer bound on the capacity region and a lower bound on the expected number of backlogged tasks in steady state. Their focus is to strike the right balance between data-locality and load-balancing to simultaneously maximize throughput and minimize delay. Naik et al. [22] proposed a novel data locality based scheduling algorithm which enhances the MapReduce framework performance in heterogeneous Hadoop cluster. The proposed scheduler dynamically divides the input data and assigns the data blocks according to the node processing capacity. It also schedules the map and reduce tasks according to the processing capacity of nodes in the heterogeneous Hadoop

cluster. Liu et al. [23] proposed that the available resources in the Hadoop MapReduce cluster are partitioned among multiple organizations who collectively fund the cluster based on computing needs. MapReduce adopts a job-level scheduling policy for a balanced distribution of tasks and effective utilization of resources. The current paper introduces a two-level query scheduling which can maximize the intra-query job-level concurrency, and at the same time speed up the query-level completion time based on the accurate prediction and queuing of queries.

Tran et al. [24] proposed a new data layout scheme that can be implemented for HDFS. The proposed layout algorithm assigns data blocks to the high-performance set and the energy-efficient set based on the data size, and keeps the replicas of data blocks in inefficient servers.

The proposed algorithm is a data distribution method, via which input data is dynamically distributed among the nodes on the base of their processing capacity in the cluster.

Recently, several scholarly papers have proposed the use of scheduling models to minimize communication costs. As an instance, an offline scheduling algorithm based on graph models was proposed by Selvitopi et al. [25], which correctly encodes the interactions between map and reduce tasks. Choi et al. [26] addressed a problem in which a map split consisted of multiple data blocks distributed and stored in different nodes. Two data-locality-aware task scheduling algorithms were proposed by Beaumont et al. [27], which optimized makespan. Makespan is defined as the time required for completing a set of jobs from the beginning to the end; i.e. the maximum completion time of all jobs. Furthermore, the scheduling algorithms proposed by Li et al. [28] were aimed at optimizing the locality-enhanced load balance and the map, local reduce, shuffle, and final reduce phases. Unlike the approaches which maximize data locality, the aim of the approach presented in the current paper is to minimize the job completion time through higher data locality rate.

Default scheduler in Hadoop MapReduce

One of the main differences between schedulers in Hadoop and other types of schedulers is different execution time of the tasks on different machines, so that based on the required amount of storage, computation, and processing power, a task may run in n time unit on a machine and in $2n$ time unit on another. However, due to the considerable data growth in data centers and the characteristics of MapReduce applications, achieving the desired goals is complicated. In Hadoop, jobs should be run by the resources provided by the cluster. Therefore, there is a specific scheduling policy for executing the jobs, according to which jobs are executed based on the available resources. In order to enhance the performance of jobs scheduling, multiple jobs can enter the cluster in the form of a batch and use system resources. To support the parallel execution of jobs, the focus should be on job scheduling mechanism based on resource sharing. There are two types of resource sharing in the MapReduce framework. In the first type, several jobs can use shared computational resources. Computational resources include CPU, memory and disk. The second type is the parallel execution of jobs, which is, in fact, the shared process; this means that a data file is processed by multiple jobs. If the number of tasks is smaller than the number of available slots, it can assign all tasks in a wave to the

free slots. However, if the number of jobs is greater than the number of available slots, then this allocation will occur in several waves.

Scheduling is regarded as an important research area in computational systems. This issue in the Hadoop is particularly important for achieving higher cluster performance. Therefore, several scheduling algorithms have been proposed for this purpose [7, 8]. Hadoop has three default schedulers: FIFO, Fair and Capacity. What follows is a brief description of FIFO and Fair algorithms and as well as their positive and negative points.

FIFO scheduler

The FIFO scheduler places applications in a queue and runs them in the order of submission. Requests for the first application in the queue are allocated first; once its requests have been satisfied, the next application in the queue is served, and so on. It will not allocate any task from other jobs if the first job in the queue still has an unassigned map task. This scheduling rule has a negative effect on the data locality, because another job's local tasks cannot be assigned to the slave node until all map tasks of the previous job are completed. Therefore, if there is a large cluster that executes many small jobs, the data locality rate could be quite low. FIFO scheduler has many limitations such as [6]: Most importantly, FIFO scheduler is not suitable for shared clusters. Large applications will use all the resources in a cluster, so each application has to wait its turn. On a shared cluster, it is better to use the Fair scheduler.

Fair scheduler

Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. The objective of the Fair scheduling algorithm is to carry out an equal distribution of computing resources among the users/jobs in the system. The scheduler actually organizes jobs by resource pool, and shares resources fairly between these pools. By default, there is a separate pool for each user. The Fair Scheduler can limit the number of concurrent running jobs per user and per pool. Moreover, it can limit the number of concurrent running tasks per pool. The fair scheduler schedules and organizes jobs into pools, where each pool has guaranteed capacity and the fair scheduler can limit concurrent running job per user as well as per pool. Jobs scheduler schedules the job using either fair sharing or first in first out and assigns the minimum share to pools. The most negative point of this algorithm is that it does not consider the job priority of each node, which could be considered an important disadvantage.

Proposed method

In the new scheduling algorithm, the focus has been placed on two aspects: dynamic jobs priority and data localization. The main ideas of this algorithm are increasing data locality rate and decreasing average completion time of map tasks. To this end, the proposed, algorithm has been named HybSMRP. The underlying ideas were inspired by two papers. In HybS algorithm, due to the fact that direct localization technique is not used, the locality rate is not high. Furthermore, in matchmaking scheduling algorithm, since dynamic job priority is not employed, when there is a large number of map tasks in the scheduling system, the response and completion

Table 1 Job profile information for experiment of Terasort and Wordcount benchmarks

Data size (GB)	Number of maps	Number of data local map task	Number of rack local map task	Average map time (s)	Average reduce time (s)	Average shuffle time(s)	Elapsed time (s)
8	65	59,61	6,4	8,9	7,52	23,26	57,103
16	130	127,129	3,1	7,8	13,93	56,58	125,205
24	195	191,192	4,3	7,7	17,150	86,83	145,321
32	260	255,255	5,5	7,8	24,198	111,124	227,447
40	325	323,322	2,3	7,8	31,242	149,155	274,508
48	390	387,378	3,12	8,9	37,288	185,234	344,652
56	455	448,449	7,6	8,9	47,353	210,256	367,780
64	520	514,493	6,27	7,10	59,403	242,342	437,904
72	585	581,575	4,10	7,9	81,451	280,350	600,987
80	650	645,639	5,11	8,9	99,511	322,413	645,1116

times are increased. Wordcount and Terasort benchmarks are used in the experiments carried out in the present paper. The result of Terasort and Wordcount are shown in Table 1 as x and y values, where x values correspond to Terasort and y values correspond to Wordcount. As shown in Table 1, with the increase in the number of maps, the run time of jobs is also increased.

The Job Tracker will launch one map Task for each map split. Typically, there is a map split for each input file. If the input file is too big, then there are two or more map splits associated with the same input file. This is the pseudocode used inside the method *get Splits()* of the *File Input Format* class, as shown in algorithm 1. This algorithm is used to determine the number of splits stored on HDFS.

Algorithm 1 Pseudocode of getSplits in HDFS

```
Input:
split_slope=1.1
split_size = dfs.blocksize
Output:
Number of splits
1: num_splits=0
2: for each input file f:
3:     remaining = f . length
4:     while(remaining / split_size > split_slope):
5:         num_splits + = 1
6:         remaining - = split_size
7:     end while
8: end for
```

The priority of a job affects the choice of the task assignment. The scheduling policies for the job priority can be assigned by an administrator by tuning the system parameters of the job's workload. HybSMRP uses dynamic priority and proportional share assignment to determine which tasks from which jobs should be assigned to the available resource node. Similar to HybS, it uses Eq. (1) to take priority of jobs based on three

parameters: runtime, waiting time and job size. The proposed algorithm ignores the order of job execution to preserve data locality.

$$Priority = \left(\frac{td}{avgWaitTime}\right)^{\propto} * \left(\frac{r}{avgRunTime}\right)^{\beta} * \left(\frac{n}{avgNumTask}\right)^{\gamma} \quad (1)$$

where *td* is the duration that a job waits in the queue, *avgWaitTime* denotes the average job waiting time in the system, *r* shows the average run time for the map task of MapReduce job, *avgRunTime* is the average run time for the map tasks for all jobs, *n* stands for the number of remaining tasks for the job and *avgNumTask* denotes the average number of tasks in all jobs in the queue. In this equation, α, β and γ are priority parameters, which can have zero or 1 values. Determining the best non zero α, β, and γ sets for a given system and workload is an interesting problem because they give applications a choice of scheduling policies. These parameters with different values can be made up of different policies, e.g. if $\alpha = 1$ and $\beta = \gamma = 0$, then this policy works like the FCFS (First Come First Serve) scheduler. With the increase in job waiting time, priority is increased; hence, the jobs which have been added to the queue earlier, get services earlier. This is similar to FCFS policy.

The waiting time is increased as the job waits in the queue, whereas the remaining size of the job is reduced when its individual tasks are completed. Moreover, if $\alpha = 0$, $\beta = -1$ and $\gamma = 0$, the scheduling policies can work like SJF (Shortest Job First). These policies were used to test the proposed algorithm.

Data Localization in HybSMRP

This section presents a short, description of the data localization technique used in the proposed scheduling algorithm. The main idea, here, is to give a fair chance to any slave nodes to get local tasks. For this purpose, one *localization ID* was used. It means that a special ID was given to any slave node for determining its status. In HybSMRP, the first point relating to jobs queue is that if it is not possible to find one local map task to assign, the scheduler will continue searching the next jobs. Second, in order to give every slave node a fair chance to get its local tasks, when a slave node fails to find a local task in the queue for the first time, no nonlocal task will be assigned to the nodes. It means that during the first heartbeat interval, no task is assigned to any of slave nodes with free map slots and is considered for local task assignment. Therefore, it results in resource wastage. In HybSMRP algorithm, in order to avoid this resource-wasting problem, one principal was used, which indicates that if one slave node fails to find a local task after two times, one nonlocal task is assigned to any slave nodes. Selection of two is to give the local tasks the opportunity to run; hence, selections greater than two cause delay in running tasks. Therefore in the proposed algorithm, local task assignment is not in series, since in addition to achieving high data locality, it also leads to high cluster utilization. In HybSMRP, a *localization ID* is assigned to any slave nodes to determine its status. According to the *ID*, the scheduler will decide whether or not to assign the node a non-local task.

Description of HybSMRP steps

Algorithm 2 provides the pseudocode of the proposed algorithm. Firstly, one *localization ID* is assigned to each node. For each heartbeat, when there are free slots on node update priority for all jobs based on Eq. (1), set the previous status of node i to localization ID[i]. Then, for each job in the job queue assign a local task to node i and decrease one slot from free slots. Afterwards, check the localization ID; if it has a null value, it will equal to 1 and continue increasing one point to it. However, if localization ID equals to zero (which may occur in case of a bug), then assign a non-local task to node *i* from the first job in the JobQueue and decrease one slot from free slots. It means that after two attempts and failures to get a local task, one non-local task is obtained. As mentioned before, this method avoids wasting resources and improves data locality rate. Finally, when a new job *j* is added into the *JobQueue,* all status of all slave nodes will be reset. It means that for each node *i* of the *N* slave nodes, *localization ID* is set to *null,* and all process will be executed again.

Algorithm 2 Pseudocode of HybSMRP algorithm

```
for each node i of the N slave nodes do
    assign  localization ID [i]=null  to all slave nodes
end for
for each heartbeat, when there are free slots on node i:
    update priority for all jobs based on eq1
    set previous status= localization ID[i]
end for
for each job in job queue do
    assign local task to node i
    if s=0 then
        break for
    else
        set s=s-1  // s is the count of free slots
        if localization ID [i]==null then
            localization ID [i]=1
        else
            localization ID [i]+=1
        end if
    end if
end for
 if localization ID [i]==0 then
    assign a non-local task to node i from the first job in the JobQueue
    set s=s-1
 end if
 when a new job j is added into the JobQueue:
    for each node i of the N slave nodes do
        set localization ID [i]=null
    end for
```

Results and discussion

The objective of the proposed scheduling algorithm is to shorten the expected response time of any given job. The proposed hybrid scheduling algorithm was evaluated with FIFO and Fair scheduling algorithm based on two parameters: map tasks' data locality rate and completion time. These parameters are the most important criteria for evaluating scheduling algorithms in Hadoop. The localization rate is calculated from Eq. (2).

$$\text{Data locality rate} = \mathrm{L}/\mathrm{N} \tag{2}$$

In this equation, L is the number of local map tasks, and N is the total number of map tasks.

Cloudera was used to test and evaluate our algorithm. Cloudera provides a prepackaged, tested and enhanced open-source distribution of the Hadoop platform. Cloudera is one of the leading innovators in Hadoop space and the largest contributor to the open-source Apache Hadoop ecosystem. In this cluster, there is one master node and 20 slave nodes. The configuration details of the cluster are shown in Table 2. Wordcount and Terasort were used as benchmarks. To evaluate the proposed HybSMRP algorithm, a submission schedule was created, which generated a submission schedule for 100 jobs. Inter-arrival times were roughly exponential with a mean of 14 s.

In this study, 6400 MB input data was generated. The block size corresponding to HDFS was considered to be 64 MB and the replication level was set as 3. TeraGen benchmark was used to generate random data. In these experiments, the cluster is always configured to have just one job queue. As already mentioned, these jobs were prioritized by Eq. 1, and this priority was based on three parameters: job size, waiting time and run time. Table 3 shows the parameters of the configuration environment for tests.

Firstly, the data locality rate was used to measure the performance of the following three schedulers: Hadoop FIFO scheduler, Hadoop Fair scheduler and HybSMRP algorithm. Table 4 shows comparisons of HybSMRP, fair and FIFO algorithms based on Data locality rate on Wordcount. In the experiment in the current study, the average locality rate percentage of FIFO is 38.4%, the average of HybSMRP is 58.7% and the average of Fair is 58.4%. From Table 4, it can be seen that the HybSMRP scheduling improves the data locality by 20.3% and 0.3%, respectively, while comparing with FIFO and Fair Scheduler on average, in the running job of Wordcount. Table 5 shows comparison of HybSMRP with FIFO and Fair algorithms based on Data locality rate on running Terasort jobs. From Table 5, it can be seen that the HybSMRP scheduling improves the data locality by 14.8% and 18.9%, respectively, while compared with FIFO and Fair Scheduler on average.

Table 2 Configuration properties of the cluster

Node	Quantity	Hardware and Hadoop configuration
Master	1	2 single-core 2.2 GHz intel-64 CPUs, 4 GB RAM, 1 Gbps Ethernet
Slave	20	2 single-core 2.2 GHz Intel-64 CPUs, 2 GB RAM, 1 Gbps Ethernet, 1 rack, 2 map and 1 reduce slots per node

Table 3 The important parameters of the configuration in the experiments

Parameters	Values
mapreduce.reduce.memory.mb	3072
mapreduce.map.memory.mb	1024
mapred.maxthreads.generate.mapoutput	2
mapreduce.tasktracker.reserved.physicalmemory.mb.low	0.95
mapred.maxthreads.partition.closer	2
mapreduce.map.sort.spill.percent	0.99
mapreduce.reduce.merge.inmem.threshold	0
mapreduce.job.reduce.slowstart.completedmaps	1
mapreduce.reduce.shuffle.parallecopies	40
mapreduce.map.speculative	False
mapreduce.reduce.speculative	False
mapreduce.map.output.compress	False
mapreduce.job.reduces	160
mapreduce.task.io.sort.mb	480
mapreduce.task.io.sort.factor	400
mfs.heapsize	35

Table 4 Numerical comparison of HybSMRP with FIFO and Fair schedulers based on data locality rate on Wordcount

Number of tasks	Percent of data locality FIFO	Percent of data locality Fair	Percent of data locality HybSMRP
10	18	60	24
20	19	62	35
30	22	65	49
40	36	66	57
50	39	69	68
60	49	60	75
70	56	55	83
80	60	50	72
90	46	49	63
100	39	48	61

Table 5 Numerical comparison of HybSMRP with FIFO and Fair schedulers based on data locality rate on Terasort

Number of tasks	Percent of data locality FIFO	Percent of data locality Fair	Percent of data locality HybSMRP
10	22	32	45
20	39	33	41
30	26	39	49
40	45	34	48
50	46	32	50
60	45	30	55
70	42	32	60
80	41	35	64
90	40	39	61
100	39	38	60

Table 6 Numerical comparison of HybSMRP with FIFO and Fair scheduler based on completion time on Wordcount

Number of tasks	Completion time (s) FIFO	Completion time (s) Fair	Completion time (s) HybSMRP
65	121	109	103
130	215	212	205
195	331	325	321
260	452	452	447
325	523	511	508
390	661	657	652
455	792	784	780
520	925	907	904
585	1005	993	987
650	1130	1121	1116

Table 7 Numerical comparison of HybSMRP with FIFO and Fair schedulers based on completion time on Terasort

Number of tasks	Completion time(s) FIFO	Completion time(s) Fair	Completion time(s) HybSMRP
65	104	75	57
130	178	136	125
195	201	187	145
260	285	248	227
325	325	298	274
390	420	384	344
455	482	387	367
520	572	498	437
585	742	645	600
650	851	734	645

In order to evaluate the completion time of HybSMRP, its performance was compared with FIFO and Fair on Wordcount and Terasort. In this test, the number of tasks was increased so that the running time of the jobs could clearly be determined. Tables 6 and 7 show the experiment results. As shown in Table 6, the proposed method is very similar to Fair Scheduler; however, HybSMRP is approximately 2.19% faster than FIFO and 0.79% faster than Fair scheduling algorithm. It can be seen from Table 7 that the HybSMRP improves completion time by about 11.51% compared to Fair and 29.15% compared to FIFO scheduling algorithm.

Conclusion and future work

In this paper, a new hybrid scheduling algorithm, HybSMRP, has been proposed in Hadoop MapReduce framework to improve data locality rate and decrease completion time with two techniques; namely dynamic priority and localization ID. The proposed algorithm was compared with Hadoop default scheduling algorithms. Experimental results demonstrate that the proposed Hybrid scheduling algorithm can often obtain

high data locality rate and low average completion time for map tasks. In the future, we will integrate our algorithm with a technique that determines the number of local tasks and mixing with delay scheduling algorithm. Delay scheduling is a simple technique for achieving locality and fairness in cluster scheduling. We also plan to focus on evaluating the performance of the proposed scheduler at scale by deploying it on a large cluster to clearly show our contribution.

Abbreviations

HybSMRP: Hybrid Scheduling MapReduce priority; HDFS: Hadoop Distributed File System; LATE: Longest Approximate Time to End; SAMR: Self-Adaptive MapReduce; CREST: Combination Re-Execution Scheduling Technology; LARTS: Locality-Aware Reduce Task Scheduler; SJF: Shortest Job First.

Acknowledgements

Not applicable.

Authors' contributions

All authors read and approved the final manuscript.

Authors' information

Abolfazl Gandomi is a Ph.D. candidate in Computer Engineering at Science and Research Branch of Islamic Azad University (SRBIAU), in Computer and Electrical Engineering Department. His main research interests include Big Data, Distributed computing systems, performance modeling and evaluation, and task scheduling algorithms.

Midia Reshadi is currently an Assistant Professor at Science and Research Branch of Islamic Azad University (SRBIAU), in Computer and Electrical Engineering Department. His research is carrying out at SRBIAU in the field of communication infrastructure and Distributed computing systems.

Ali Movaghar is a Professor in the Department of Computer Engineering at Sharif University of Technology in Tehran. His research interests include performance/dependability modeling and formal verification of wireless networks, distributed real-time systems and Big Data. He is a senior member of the IEEE and the ACM.

Ahmad Khademzadeh is a Professor in the Digital Communication and Information Theory and Big Data. He is currently the Head of Education and National Scientific and Informational Scientific Cooperation Department at Iran Telecom Research Center (ITRC). Dr. Khademzadeh has been received four distinguished national and international awards including Kharazmi International Award, and has been selected as the National outstanding researcher of the Iran Ministry of Information and Communication Technology.

Funding

No funding has been received for the conduct of this work and preparation of this manuscript.

Availability of data and materials

If anyone is interested in our work, we are ready to provide more details about the Hadoop application code source.

Competing interests

The authors declare that they have no competing interests.

Author details

[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran. [2] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. [3] Iran Telecommunication Research Center, ITRC, Tehran, Iran.

Received: 21 May 2019 Accepted: 27 September 2019
Published online: 30 November 2019

References

1. Dittrich J, Quiané-Ruiz JA. Efficient big data processing in Hadoop, MapReduce. Proceedings of the VLDB Endowment. 2012;5(12):2014–5. https://doi.org/10.14778/2367502.2367562.
2. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13. https://doi.org/10.1145/1327452.1327492.
3. Babu S. Towards automatic optimization of MapReduce programs. In: Proceedings of the 1st ACM symposium on Cloud computing. 2010; p. 137–142. http://dx.doi.org/10.1145/1807128.1807150.
4. Lee KH, Lee YJ, Choi H, Chung YD, Moon B. Parallel data processing with MapReduce: a survey. ACM SIGMOD Record. 2012;40(4):11–20. https://doi.org/10.1145/2094114.2094118.
5. Bu X, Rao J, Xu CZ. Interference and locality-aware task scheduling for MapReduce applications in virtual clusters. In: Proceedings of the 22nd international symposium on High-performance parallel and distributed computing. 2013; p. 227–238. http://dx.doi.org/10.1145/2493123.2462904.
6. Polato I, Ré R, Goldman A, Kon F. A comprehensive view of Hadoop research—a systematic literature review. J Netw Comput Appl. 2014;46:1–25. https://doi.org/10.1016/j.jnca.2014.07.022.

7. T White (2015) Hadoop: The definitive guide. O'Reilly Media, Inc.
8. Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling In Proceedings of the 5th European conference on Computer systems. 2010; p. 265–278. http://dx.doi.org/10.1145/1755913.1755940.
9. He C, Lu Y, Swanson D. Matchmaking: a new MapReduce scheduling technique. In: IEEE third international conference on cloud computing technology and science (CloudCom), (2011). 2011; p. 40–47. https://doi.org/10.1109/cloudcom.2011.16.
10. Nguyen P, Simon T, Halem M, Chapman D, Le Q. A hybrid scheduling algorithm for data intensive workloads in a MapReduce environment. In: Proceedings of the 2012 IEEE/ACM fifth international conference on utility and cloud computing. 2012; p 161–167. http://dx.doi.org/10.1109/UCC.2012.32.
11. Nandakumar AN, Nandita Y. A survey on data mining algorithms on Apache Hadoop platform. Int J Emerg Technol Adv Eng. 2014;4(1):563–5.
12. Miner D, Shook A. MapReduce design patterns: building effective algorithms and analytics for Hadoop and other systems. Sebastopol: O'Reilly Media, Inc.; 2012.
13. Holmes A. Hadoop in practice. Shelter Island: Manning Publications Co.; 2012.
14. Khan M, Jin Y, Li M, Xiang Y, Jiang C. Hadoop performance modeling for job estimation and resource provisioning. IEEE Trans Parallel Distrib Syst. 2016;27(2):441–54. https://doi.org/10.1109/TPDS.2015.2405552.
15. Wang K, Zhou X, Li T, Zhao D, Lang M, Raicu I. Optimizing load balancing and data-locality with data-aware scheduling. In 2014 IEEE international conference on Big Data (Big Data). 2014; p. 119–128. http://dx.doi.org/10.1109/BigData.2014.7004220.
16. Zaharia M, Konwinski A, Joseph AD, Katz RH, Stoica I. Improving MapReduce performance in heterogeneous environments. OSDI. 2008;8(4):7.
17. Chen Q, Zhang D, Guo M, Deng Q, Guo S. SAMR: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In: IEEE 10th international conference on computer and information technology (CIT). 2010; p 2736–2743. http://dx.doi.org/10.1109/CIT.2010.458.
18. Lei L, Wo T, Hu C. CREST: towards fast speculation of straggler tasks in MapReduce. In: IEEE 8th international conference on e-business engineering (ICEBE). 2011; p. 311–316. http://dx.doi.org/10.1109/ICEBE.2011.37.
19. Hammoud M, Sakr MF. Locality-aware reduce task scheduling for MapReduce. In: IEEE third international conference on cloud computing technology and science (CloudCom), 2011. 2011; p. 570–576. http://dx.doi.org/10.1109/CloudCom.2011.87.
20. Ibrahim S, Jin H, Lu L, He B, Antoniu G, Wu S. Maestro: replica-aware map scheduling for MapReduce. In: Proceedings of the 2012 12th IEEE/ACM international symposium on cluster, cloud and grid computing (ccgrid 2012). 2012; p. 435–442. http://dx.doi.org/10.1109/CCGrid.2012.122.
21. Wang W, Zhu K, Ying L, Tan J, Zhang L. Map task scheduling in MapReduce with data locality: throughput and heavy-traffic optimality. IEEE/ACM Trans Netw. 2016;24(1):190–203. https://doi.org/10.1109/TNET.2014.2362745.
22. Naik NS, Negi A, Tapas Bapu BR, Anitha R. A data locality based scheduler to enhance MapReduce performance in heterogeneous environments. Future Gener Comput Syst. 2019;90:423–34. https://doi.org/10.1016/j.future.2018.07.043.
23. Liu Z, Nath AK, Ding X, Fu H, Khan M, Yu W. Multivariate modeling and two-level scheduling of analytic queries. Parallel Comput. 2019. https://doi.org/10.1016/j.parco.2019.01.006.
24. Tran XT, Van Do T, Rotter C, Wang D. A new data layout scheme for energy-efficient MapReduce processing tasks. J Grid Comput. 2018;16(2):285–98. https://doi.org/10.1007/s10723-018-9433-7.
25. Selvitopi O, Demirci GV, Turk A, Aykanat C. Locality-aware and load-balanced static task scheduling for MapReduce. Future Gener Comput Syst. 2019;90:49–61. https://doi.org/10.1016/j.future.2018.06.035.
26. Choi D, Jeon M, Kim N, Lee BD. An enhanced data-locality-aware task scheduling algorithm for hadoop applications. IEEE Syst J. 2017;12(4):3346–57. https://doi.org/10.1109/JSYST.2017.2764481.
27. Beaumont O, Lambert T, Marchal L, Thomas B. Data-locality aware dynamic schedulers for independent tasks with replicated inputs. In: 2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW). 2018; p. 1206–1213. https://doi.org/10.1109/IPDPSW.2018.00187.
28. Li J, Wang J, Lyu B, Wu J, Yang X. An improved algorithm for optimizing MapReduce based on locality and overlapping. Tsinghua Sci Technol. 2018;23(6):744–53. https://doi.org/10.26599/TST.2018.9010115.

Publisher's Note