J Grid Computing (2018) 16:535–551
https://doi.org/10.1007/s10723-018-9464-0

A Task-Based Greedy Scheduling Algorithm for Minimizing Energy of MapReduce Jobs

Mostafa Hadadian Nejad Yousefi ·
Maziar Goudarzi

Received: 21 August 2017 / Accepted: 7 August 2018 / Published online: 22 August 2018

Abstract MapReduce and its open source implementation, Hadoop, have gained widespread adoption for parallel processing of big data jobs. Since the number of such big data jobs is also rapidly rising, reducing their energy consumption is increasingly more important to reduce environmental impact as well as operational costs. Prior work by Mashayekhy et al. (IEEE Trans. Parallel Distributed Syst. **26**, 2720–2733, 2016), has tackled the problem of energy-aware scheduling of a single MapReduce job but we provide a far more efficient heuristic in this paper. We first model the problem as an Integer Linear Program to find the optimal solution using ILP solvers. Then we present a task-based greedy scheduling algorithm, TGSAVE, to select a slot for each task to minimize the total energy consumption of the MapReduce job for big data applications in heterogeneous environments without significant performance loss while satisfying the service level agreement (SLA). We perform several experiments on a Hadoop cluster to measure characteristics of tasks for nine different applications to evaluate our proposed algorithm. The results show that the total energy consumption of MapReduce jobs obtained by TGSAVE is up to 35% less than that achieved by EMRSA proposed in Mashayekhy et al. (IEEE Trans. Parallel Distributed Syst. **26**, 2720–2733, 2016), its closest rival, for same workloads. Besides, TGSAVE is capable of finding a solution in same order of time for up to 74% tighter deadlines than the tightest deadline that EMRSA can find a feasible one. On average, TGSAVE solution is approximately 1.4% far from the optimal solution, and it can meet deadlines as tight as 12%, on average, above the energy-oblivious minimum makespan in the benchmarks we examined.

Keywords Big data · Energy-aware · MapReduce · Scheduling · Heterogeneous systems

1 Introduction

We are stepping into the Digital Transformation era. This is not only about improving the functions but also making new ones. Digitalization inherently makes new opportunities in every domain. Every business must embrace the change to be competitive. Data which used to be stored on papers is now going to be stored, processed, and transferred digitally. The rapid growth of data sources produces an enormous amount of data. We need new techniques like distributed processing to bring gold out of this big amount of raw data while reducing the energy consumed.

The global data center traffic is forecasted to triple from 2014 to 2019. It is also forecasted that by 2019,

M. H. N. Yousefi · M. Goudarzi (✉)
Computer Engineering Department, Sharif University of Technology, Tehran, Iran
e-mail: hadadian@ce.sharif.edu

M. Goudarzi
e-mail: goudarzi@sharif.edu

the majority of workloads will have been processed by data centers using popular models such as MapReduce [12]. Hence, more resources including energy will be required to process this increasingly soaring amount of data. Meanwhile, the energy production has not significantly grown in recent years [1]. Therefore, processing of this data will soon meet an upper bound for power consumption. In consequence, it is important to reduce the energy of MapReduce jobs in data centers.

Improving energy efficiency of such IT systems can be done through hardware as well as software. Software approaches have a better opportunity for innovation since it is faster to implement, more flexible, and more adaptable to hardware needs and specifications. Thus, we focus on such opportunities in this paper.

MapReduce is a platform for parallel processing of large datasets [13]. It can reduce cost since it has the capability to be run on clusters of cheap commodity machines instead of expensive specialized machines. Moreover, MapReduce is highly scalable. It can process petabytes of data on clusters of thousands or even more machines. These features cause MapReduce Platform to become a popular platform for data centers and warehouse-scale computers.

Hadoop [7] is an open-source implementation of MapReduce. Popular schedulers of the current version of Hadoop are FIFO (First In First Out), Fair Scheduler [8], Capacity Scheduler [6] and HOD (Hadoop On Demand) Scheduler [9]. None of them pays attention to energy efficiency. Fair Scheduler assigns resources to jobs such that all jobs get, on average, an equal share of resources over time. Capacity Scheduler is designed to run Hadoop Map-Reduce as a multi-tenant, shared cluster in an while maximizing the utilization and the throughput of the cluster when running Map-Reduce applications. HOD Scheduler is a system for provisioning and managing independent Hadoop MapReduce and Hadoop Distributed File System (HDFS) instances on a shared cluster of nodes. The HOD approach uses the Torque resource manager [2] for node allocation based on the needs of the virtual cluster. In this paper, we present a greedy scheduling algorithm for minimizing total energy of MapReduce jobs for big data applications in heterogeneous environments without significant performance loss while satisfying Service Level Agreement (SLA). Mashayekhy et al. [23] have also provided two heuristics for the same problem, but we push state of the art one step further by providing a more efficient algorithm. Similar to EMRSA, our work is also designed for scheduling single jobs. However, it supports multiple jobs by executing them one after the other.

In almost every data center, there is some sort of job, which should be not only run at a regular interval but also done before a deadline. Minimizing this type of jobs are the main focus (but not limited to) of our work. For example, a mail service has a spam detection job that should be run every 10 minutes, with a deadline of 5 minutes or a health-care company may need to collect data of its patients from sensors every 5 minutes and process them in 2 minutes. There are many other examples of such repetitive jobs. Interval and deadline can vary from minutes to hours or even days for different applications. Since these sorts of jobs are typically repetitive, job profiling can be used to estimate performance characteristics of them, such as energy consumption and processing time as also suggested and used in [23]. A scheduler can use this data and schedule jobs for future runs. As these jobs run so many times, profiling for the first run does not cost much on average. Thus, similar to [23], our algorithm also uses pre-computed characteristic data of jobs to minimize the total energy of future jobs while satisfying the SLA.

1.1 Our Contribution

To the best of our knowledge, Mashayekhy et al. [23] and their EMRSA algorithm, represent the prior state of the art to solve the problem of task assignment and scheduling of a single MapReduce job for energy-efficient computation on a heterogeneous cluster of computers. We analyze the prior art deeply in Section 2 to find the opportunities to improve the solution of this problem. We enhance this state of the art and compare our solution with EMRSA and the optimal one. Our major contributions are described as below:

- We first model the problem as an integer linear program which is easier to understand and faster to execute than the LP model presented in [23].
- We provide our greedy algorithm that uses a heuristic function to solve the problem. Decision making in the optimization procedure is based on minimizing the energy consumption of tasks,

by prioritizing the tasks with a higher variance of energy consumption over given machines and then selecting the best slot for each one of them. We called our proposed algorithm TGSAVE, where TGSAVE is an acronym for Task-based Greedy Scheduling Algorithm based on Variance to minimize Energy consumption of MapReduce.
- We further analyze the complexity of our algorithm and show that the time complexity of our algorithm is polynomial in the number of map slots, reduce slots, map tasks and reduce tasks, and hence, can work efficiently for enormous problem sizes.
- We performed several experiments on a Hadoop cluster with different machines to measure the energy consumption and processing time of several MapReduce benchmark applications covering different characteristics including Word-Count, Sequence-Count, Self-Join, Ranked-Inverted-Index, Inverted-Index, Histogram-Movies, Classification, Adjacency-List, K-Means, and Page-Rank. We use this data as well as the same Tera Sort benchmark data used for evaluating EMRSA in [23] as input data for our designed algorithm to analyze the performance.
- TGSAVE saves energy up to 35% more than EMSRA and is at most only 4% (1.4% on average) far from the optimal solution.
- TGSAVE finds solutions under deadlines up to 74% tighter than the tightest one feasible by EMRSA and it can meet deadlines as tight as only 12%, on average, bigger than the energy-oblivious minimum makespan.
- With all above improvements, TGSAVE still works in the same order of time as EMRSA.

1.2 Organization

The rest of the paper is organized as follows. In Section 2, the motivational example is presented. In Section 3, we review the related work in this field. In Section 4, we describe the problem of scheduling a MapReduce job in heterogeneous environments for big data application and model the problem as an integer linear program. In Section 5, we present our proposed algorithm. In Section 6, we present the experimental results. Finally, we conclude the paper and present possible directions for future work in Section 7.

2 Motivational Example

Mashayekhy et al. [23] proposed, EMRSA, an energy-aware scheduler of a MapReduce job for big data applications in heterogeneous environments. EMRSA uses prior knowledge about workload to estimate future response time and energy consumption of tasks.

We analyzed the performance of EMRSA and Optimal algorithms through a number of experiments run on our own setup. The details of the workloads and the experimental setup are described in Sections 6.1 and 6.2 respectively. The results show that EMRSA performance is up to 36% far from the Optimal choice. More importantly, for each task assignment, the distance between EMRSA and Optimal increases when the variance of energy consumption of the task on different machines increases.

We analyzed task assignment of EMRSA and Optimal solutions. The results are shown in Fig. 1. The horizontal axis represents tasks in ascending order (from left to right) of above variance. The primary vertical axis (left) shows the EMRSA/Optimal energy consumption for each task. The secondary vertical axis illustrates the variance of energy consumption on different machines for each task. The results of our evaluation show that in EMRSA solution a task with a higher variance is often farther from the Optimal choice. Which means the gap between EMRSA and the Optimal solutions grows with the increase in the level of heterogeneity of the cluster. This is a core observation of ours and represents the bottom line underlying our proposed algorithm. Based on the aforementioned observation, we decided to give a task with a higher variance in energy consumption over machines a higher priority during assignment so that it is assigned to the machine that can execute it more efficiently.

Thus, the core idea behind our algorithm is to first carefully select tasks for assignment (i.e., start from those with higher variance) and then choose the best available machine for them. This is in contrast to EMRSA which starts from more efficient machines based on their average energy for all tasks, and then chooses tasks to run on them. Experimental results show that our solution is much more effective and is at most 4.6% far from Optimal. The details are presented in Sections 5 and 6.

The work by Mashayekhy et al. in [23] is the closest one to ours and the basis for comparison for us. Both

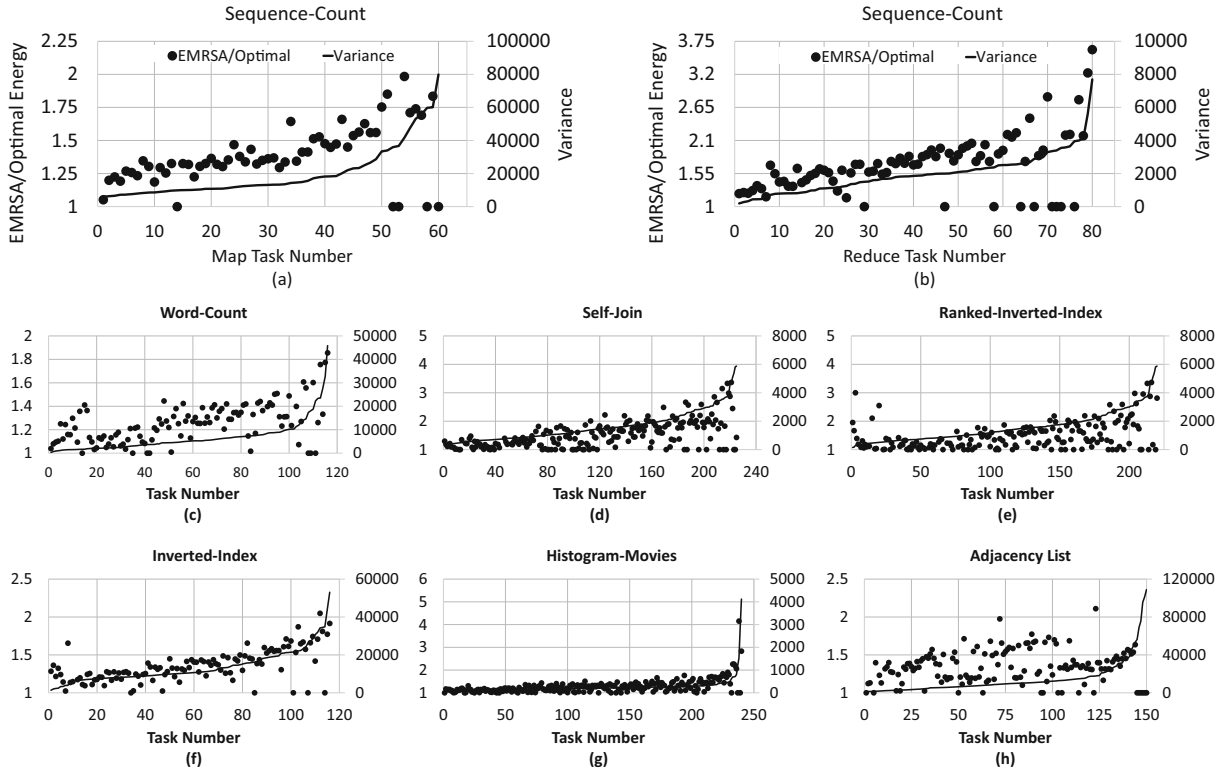

Fig. 1 EMRSA to the Optimal ratio of energy consumption of each task for PUMA benchmarks. Tasks are sorted by the variance of their energy consumption on available machines. Sequence count map tasks (**a**), and reduce tasks (**b**) are shown separately to view more details, while for the (**c**-**h**) benchmarks, both map as well as reduce tasks are shown together. The EMRSA/Optimal values are drawn as Dots on primary axis, and the Variances are shown as a line on the secondary axis

works tackle the same problem, but the solutions are different. EMRSA schedules tasks with a slot-based perspective. In the first step, EMRSA determines the most energy-efficient map slot and reduce slot, then assigns map and reduce tasks with longest processing time to these two slots until no other task can be assigned; afterward, it sets the deadline for executing all map tasks based on the processing time of the selected tasks on these two slots. In the second step, EMRSA continues to select the next most energy-efficient slots among the remaining slots and fills them with tasks that have the longest processing time on the selected slots until all tasks are assigned. On the contrary, our proposed algorithm schedules tasks with a task-based perspective. In the first step, our proposed algorithm determines a feasible deadline for map tasks based on the average processing time of each task on every slot. In the second step, TGSAVE prioritizes the tasks with the highest variance of energy consumption on available slots since this means such tasks have the greatest effect on total energy consumption. Thus, starting from the top of the above sorted list of tasks, the first available slot with minimum energy consumption is chosen for it while considering the map as well as final deadlines (or equivalently, while satisfying a given SLA), until all tasks are assigned.

3 Related Work

There are many prior works on hardware-based energy reduction techniques such as turning off a subset of machines [14, 17, 20, 21] or putting them into low power mode [24], memory management [10, 18, 22, 27, 31], dynamic voltage frequency scaling [29, 30] and using proper processor for each workload [39]. Meanwhile, software-based approaches efficiently use IT equipment components to reduce energy consumption. Many prior studies focus on performance.

Cho et al. [11] designed Natjam, a system that supports arbitrary job priorities and hard real-time scheduling. Natjam improves completion time and Natjam-R, an extension of Natjam, satisfies more job deadlines. Wolf et al. [35] described FLEX as a flexible scheduling scheme with the goal of optimizing a variety of standard scheduling theory metrics including response time and makespan. The FLEX allocation scheduler can achieve performance close to optimal.

The whole point of Heterogeneous Computing is to use the right processor, in the right place, at the right time. Heterogeneity provides a good opportunity for researchers to push their studies forward in order to gain better performance, lower cost, and energy efficiency in data centers. Many studies are working on improving the performance of MapReduce in heterogeneous environments. Ahmad et al. [3] addressed two key reasons for poor performance of MapReduce on heterogeneous clusters. They also proposed Tarazu, a communication-aware suite of optimizations to improve MapReduce performance on heterogeneous clusters. Krish et al. [19] proposed a hardware-aware scheduler to improve the resource-application match. Zaharia et al. [40] designed a scheduling algorithm, LATE, that is robust to heterogeneity while improving response time. Yang and Chen [38] designed an adaptive task allocation scheduler, ATAS, to improve LATE scheduler. There are also some other studies with the goal of improving the performance of MapReduce in heterogeneous environments with paying little or no attention to energy efficiency [5, 16, 28, 32, 33, 36].

After reaching a power limit because of various reasons such as insufficient electricity supply and cost budget limitation, recent research started to pay attention to energy-efficient computing. Yan et al. [37] present a MapReduce job scheduler for the heterogeneous multi-core processor, called DyScale. The goal of DyScale is to improve the performance of MapReduce in heterogeneous environments while satisfying a given power budget. The difference between DyScale and our proposed work is that our goal is minimizing the energy consumption of MapReduce while meeting a given deadline to finish all tasks. Zhang et al. [41] presented a heterogeneity-aware framework, called HARMONY, that dynamically adjusts the number of machines to make a compromise between energy saving and scheduling delay. It also considers the reconfiguration cost.

4 Problem Description and Model

MapReduce is an abstraction to organize massively parallel tasks. Hadoop is an open-source implementation of it. Hadoop is a software framework for distributed processing of large data sets over several machines. The main idea of MapReduce could be summarized as mapping data set into a collection of key and value pairs and then reducing all pairs with the same key. A MapReduce job includes a set of map and reduce tasks, distributed over slave nodes to be executed. The first-order execution of a MapReduce job can be divided into two phases: Map phase and Reduce phase, executing map tasks and reduce tasks respectively. Reduce phase typically starts after execution of all map tasks are over. The master node in distributed Hadoop clusters hosts various storage and processing management services. One of master node services, JobTracker, performs scheduling of the jobs and deploying them to the TaskTracker nodes. An instance of the TaskTracker daemon operates on every slave node in the Hadoop cluster. Every TaskTracker is configured with a set of slots. These indicate the number of tasks that can be simultaneously accepted from JobTracker. Slots are also divided into two type of map slots and reduce slots, such that each map task must be assigned to a map slot, and each reduce task must be assigned to a reduce slot. To run a MapReduce workflow, one needs to create two scripts: the map script, and the reduce script. The framework will handle the rest automatically. First, the framework splits the input data into smaller segments and passes each segment to a distinct machine. Afterward, each machine runs the map script on the portion of data assigned to it. The map script takes some input data and maps it to a number of $< key, value >$ pairs. Then, emits them as intermediate data. The reduce script takes these intermediate data as a collection of $< key, value >$ pairs and reduces the pairs with the same key into a single pair.

The problem is to schedule map and reduce tasks and assign each of them to a slot for minimizing the overall energy consumption of all tasks for a given deadline in heterogeneous environments. Scheduling is based on prior knowledge about the workload,

including estimated energy consumption and estimated processing time of each task while executing on each machine. In fact, assigning a task to a slot means placing data on the machine to which the slot belongs, and running the corresponding script.

It is noteworthy that actual MapReduce execution is more complicated than the above mentioned abstraction and indeed involves a number of other steps and optimization opportunities as well. We will discuss a number of them in Section 6.4. For a fair comparison to the prior art, however, in this work we suffice to this abstraction as proposed in [23] which serves as the state of the art to the best of our knowledge, and hence, the basis for comparison. Further extensions are discussed in Section 7.

We consider a MapReduce job comprised of a set of map tasks and reduce tasks. Let i be the ID number of a map task and j be the ID number of a reduce task. In Hadoop, the number of map tasks is determined by the total size of inputs; i.e., the total number of blocks of the input files while the block size is configurable per file. The number of reduce tasks for the job is set by the user. We also have a number of map slots and a number of reduce slots available on heterogeneous machines for executing the respective tasks. Let k be the ID number of a map slot and l be the ID number of a reduce slot. The system administrator configures the number of slots on each machine when the Hadoop cluster is setup. Let I and J be the set of map and reduce tasks, and K and L be the set of map and reduce slots, respectively. Let D be the given deadline for completing all tasks before missing it. Each slot executes tasks in sequential order but different slots can work simultaneously. Similar to [23], we also assume that the reduce slots can work only when all map slots have done their tasks. We denote D^m and D^r as deadlines (relative to the corresponding start times) for completing all of map tasks and all of reduce tasks, respectively. We consider the problem in heterogeneous environments. Therefore, processing time and energy consumption of each task, either map task or reduce task, can vary per machine. As inputs of our problem, let E^m_{ik} and T^m_{ik} be the estimated energy consumption and the estimated processing time of map task i while executing on map slot k, respectively. Similarly, let E^r_{jl} and T^r_{jl} be the estimated energy consumption and the estimated processing time of reduce task j while executing on reduce slot l, respectively. Note that E^m_{ik}, T^m_{ik}, E^r_{jl} and T^r_{jl} are elements of tables E^m, T^m, E^r and T^r respectively. Further, note that our problem is to schedule jobs that repetitively run on the machines; jobs can even be run periodically every day. Thus, it is feasible to gather the needed information about the workload by one-time profiling and make estimations even better in every run of the job.

We model the problem as an integer linear program. In order to do so, we define two sets of variables for deciding which slot should run which subset of tasks. We denote M_{ik} and R_{jl} as follows:

$$M_{ik} = \begin{cases} 1 & \text{if map task } i \text{ is assigned to map slot } k \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

$$R_{jl} = \begin{cases} 1 & \text{if reduce task } j \text{ is assigned to reduce slot } l \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

The objective function is as follow:

$$Min \sum_{i \in I} \sum_{k \in K} E^m_{ik} M_{ik} \sum_{j \in J} \sum_{l \in L} E^r_{jl} M_{jl} \tag{3}$$

The constraints are described as follow:

$$\sum_{k \in K} M_{ik} = 1, \forall i \in I \tag{4}$$

$$\sum_{l \in L} R_{jl} = 1, \forall j \in J \tag{5}$$

$$D^m + D^r = D \tag{6}$$

$$\sum_{i \in I} T^m_{ik} M_{ik} \leq D^m, \forall k \in K \tag{7}$$

$$\sum_{j \in J} T^r_{jl} M_{jl} \leq D^r, \forall l \in L \tag{8}$$

The objective function (3) is to minimize total energy consumption of all tasks. Constraint (4) ensures that each map task is assigned to one and only one map slot, and all task must be assigned. Similar to constraint (4), constraint (5) ensures the same for reduce tasks. Constraint (6) guarantees that non-overlapping execution of map and reduce phase will not miss the deadline. Constraint (7) ensures that

map slots complete execution of all map tasks before missing D^m. Constraint (8) ensure that reduce slots complete execution of all reduce task before missing D^r.

Note that, as the objective of the problem is to minimize the sum of energy consumption of map tasks and reduce tasks, and since scheduling is strongly related to the deadline, choosing D^m and D^r is very important. Relaxing D^m corresponds to more opportunity to reduce the energy consumption of map tasks while at the same time reducing that opportunity for reduce tasks. Clearly, there is a trade-off between energy consumption of map tasks and reduce tasks, and making the right balance is the key for approaching the optimal solution of the problem. It is noteworthy that both our model and the ILP model presented in [23], model the same problem, but our above formulation is simpler to solve, due to reducing the number of required variables, and easier to understand.

5 Greedy Scheduling Algorithm

We analyze EMRSA deeply as desribed in section 2 and found its weaknesses as below:

- EMRSA sets the map phase deadline only based on the knowledge about the most energy efficient machine. In heterogeneous environments, the best machine can be an outlier. Thus, such choice of map phase deadline may not be a good one for every other machine in the cluster.
- EMRSA's underlying assumption is that if a task has a smaller processing time on a machine, the machine consumes more energy to execute the task faster. As also described in [26], this assumption is not always true as witnessed in newer computers that consume less while processing more.
- EMRSA selects a machine, fills it with tasks, and then selects next machine to fill. Therefore, it may lose some opportunities to minimize the energy consumption.
- EMRSA assigns tasks to machines only based on the processing time. It does not consider the energy consumption of tasks.

Consider two types of operation, A and B, and two machines, X and Y. Machine X (Y) executes type A (B) operations faster and with less energy. EMRSA selects one of them and fills it with type A and B operations while the other machine is idle and can execute one type of operation better. As a result, if we give EMRSA a larger deadline, it may lose more and more opportunities to minimize the total energy consumption, which is obviously undesirable. Based on this analysis, we designed our proposed algorithm that takes knowledge about every machine and every task into account for scheduling a single MapReduce job in every heterogeneous environment.

Our algorithm has three main phases. The first phase is to estimate the map and reduce phases deadline, the second phase is to set up queues, and the final phase is to assign each task to a machine while satisfying the SLA. Function 1 shows this top-level procedure, where ESTIMATE, SETUPQ, and ASSIGN are functions for doing first, second and third phase respectively.

Function 1 TGSAVE main function

```
1 ESTIMATE();
2 SETUPQ();
3 ASSIGN();
```

In the first phase as shown in Function 2, TGSAVE reads inputs of the problem as described in Section 4. Then based on the knowledge about the workload, it determines D^m and D^r. Determining deadline for map phase and reduce phase requires two vectors $\overline{T}^m$ and $\overline{T}^r$. Let $\overline{T}^m_i$ be the average processing time of map task i on every map slot (line 2). Similarly, let $\overline{T}^r_j$ be the average processing time of reduce task j on every reduce slot (line 2). Line 4 determines the deadline for map tasks, D^m, proportional to the sum of these average execution times of tasks. Line 5 sets the remaining time of Deadline to D^r. Also, for our heuristic function to later select tasks based on their priority, we need two vectors $\overline{E}^m$ and $\overline{E}^r$, which are defined similar to above $\overline{T}$ vectors (line 3).

In the second phase, TGSAVE setups required queues for scheduling as shown in Function 3. We need two priority queues, one to sort map tasks and one for reduce tasks. In addition, we need a metric for deciding which task should be assigned earlier. We use the variance of energy consumptions of each task over all machines as this metric. The higher the

Function 2 ESTIMATE

1 read input (D, T^m, T^r, E^m, E^r)
2 $\overline{T}_i^m = \frac{\sum_{k\in K} T_{ik}^m}{|K|}$, $\overline{T}_j^r = \frac{\sum_{l\in L} T_{jl}^r}{|L|}$
3 $\overline{E}_i^m = \frac{\sum_{k\in K} E_{ik}^m}{|K|}$, $\overline{E}_j^r = \frac{\sum_{l\in L} E_{jl}^r}{|L|}$
4 $D_m = D \times \frac{\sum_{i\in I} \overline{T}_i^m}{\sum_{i\in I} \overline{T}_i^m + \sum_{j\in J} \overline{T}_j^r}$
5 $D^r = D - D^m$

variance, the higher the priority. Tasks with greater variance are more important for scheduling because energy consumption of running the task can vary a lot from a machine to another machine. For example, for a task with zero variance, it does not matter to which machine it is assigned because energy consumptions of the task on every machine are the same. On the other extreme, a large variance means there is a large difference between the choices. We add tasks and their priorities in respective priority queues to sort them.

Let Q^m and Q^r be the priority queues based on the above variances in energy consumption for map tasks and reduce tasks, respectively (lines 1-5). Also, every task needs a priority queue to select a proper slot. Let S_i^m and S_j^r be the priority queue of slots for map task i and reduce task j, where slots are prioritized based on the energy consumption (lines 6-12).

Function 3 SETUPQ

1 create priority queues of tasks Q^m and Q^r
2 for every map task i do
3 $\quad Q^m.put(i, \frac{\sum_{k\in K}(E_{ik}^m - \overline{E}_i^m)^2}{|K|})$
4 for every reduce task j do
5 $\quad Q^r.put(j, \frac{\sum_{l\in L}(E_{jl}^m - \overline{E}_j^m)^2}{|L|})$
6 create priority queues of slots for each map task S_i^m and each reduce task S_j^r
7 for every map task i do
8 $\quad$ for every map slot k do
9 $\quad\quad S_i^m.put(k, E_{ik}^m)$
10 for every reduce task j do
11 $\quad$ for every reduce slot l do
12 $\quad\quad S_j^r.put(l, E_{jl}^r)$

The final phase is to assign tasks to respective slots, as shown in Algorithm 4. We need to know the running time of each slot to ensure satisfying the deadline. We define two vectors B^m and B^r, where B_k^m and B_l^r are the busy time of map slot k and reduce slot l, respectively (line 1). We separate our problem into two independent subproblems by determining the deadline for completing map and reduce tasks. The objective of this phase is to minimize each subproblem separately. Therefore, in this phase, we first assign map tasks to map slots (lines 3-15). After all map tasks are assigned, TGSAVE computes how much of the time for completing map tasks is still unused and adds this amount of time to the deadline of completing reduce tasks (line 16) which may bring better opportunity for reducing the energy consumption of reduce tasks. Then, we assign reduce tasks while satisfying determined deadline (line 17 to 29). We could have first assigned reduce tasks to reduce slots while satisfying deadline for executing reduce tasks, and then assign map tasks while satisfying map phase deadline plus the surplus amount of time remained from assigning reduce tasks. Our experiments over a number of workloads show that on average, the performances of both strategies are almost the same. However, we can run both strategies and mark the best result as the outcome of the algorithm. Note that in such case, the processing time of the algorithm less than doubles since some computations are shared between the two runs. The output of the ASSIGN() function, as well as the top level algorithm, is tables of M and R that contains the information about how tasks are assigned to slots.

For assigning tasks, TGSAVE selects tasks one after another by removing them from the respective queue (lines 3-5, and also 17-19). The pollMax function (lines 4 and 18) selects the task that has the highest variance among unassigned tasks. After a task is selected, we remove slots one by one from the respective queue (lines 6-7, and also 20-21). The pollMin function removes the slot with the lowest energy consumption among all slots (lines 7 and 21). We check slots for finding the slot such that if the selected task executes on the slot, it has the smallest energy consumption while satisfying the deadline (lines 8 and 22). When such slot is found, we stop the search, assign the task to the slot, update busy time of selected slot (lines 9-12, and also 23-26) and select next task until no task is left unassigned, or cannot find a solution (lines 13-14, and also 27-28). If we find a feasible schedule, then return M and R as the output of the algorithm that shows which slot should run which subset of tasks (line 30).

Algorithm 4 ASSIGN

1 $B^m = \{0\}$, $B^r = \{0\}$ //Busy-time of map and reduce slots
2 $M = \{0\}$, $R = \{0\}$ // Task-to-slot binary variables for map and reduce tasks
3 while Q^m is not empty do
4 $\quad i = Q^m.pollMax()$
5 $\quad$ isComplete = false
6 $\quad$ while S_i^m is not empty do
7 $\quad\quad k = S_i^m.pollMin()$
8 $\quad\quad$ if $B_k^m + T_{ik}^m \leq D^m$ then
9 $\quad\quad\quad B_k^m = B_k^m + T_{ik}^m$
10 $\quad\quad\quad M_{ik} = 1$
11 $\quad\quad\quad$ isComplete = true
12 $\quad\quad\quad$ break
13 if isComplete = false then
14 $\quad$ Output: algorithm fails to find a feasible schedule
15 $\quad$ return
16 $D^r = D^r + (D^m - \max_{k \in K}(B_k^m))$
17 while Q^r is not empty do
18 $\quad j = Q^r.pollMax()$
19 $\quad$ isComplete = false
20 $\quad$ while S_j^r is not empty do
21 $\quad\quad l = S_j^r.pollMin()$
22 $\quad\quad$ if $B_l^r + T_{jl}^r \leq D^r$ then
23 $\quad\quad\quad B_l^r = B_l^r + T_{jl}^r$
24 $\quad\quad\quad R_{jl} = 1$
25 $\quad\quad\quad$ isComplete = true
26 $\quad\quad\quad$ break
27 if isComplete = false then
28 $\quad$ Output: algorithm fails to find a feasible schedule
29 $\quad$ return
30 Output: M and R

5.1 Complexity Analysis

In Function 2, time complexity of reading inputs is $O(\mid I \mid\mid K \mid + \mid J \mid\mid L \mid)$ where $\mid I \mid$, $\mid K \mid$, $\mid J \mid$, and $\mid L \mid$ are numbers of map tasks, number of map slots, number of reduce tasks and number of reduce slots, respectively (Function 2, lines 1). Similarly, Time Complexity of computation in lines 2-3 is $O(\mid I \mid\mid K \mid + \mid J \mid\mid L \mid)$. That of determining deadline in line 4 is $O(\mid I \mid + \mid J \mid)$. Thus, total time complexity of Function 2 is $O(\mid I \mid\mid K \mid + \mid J \mid\mid L \mid)$.

We use Fibonacci heap [15] for implementation of priority queues. Time complexities of insertion and deletion are $O(1)$ and $O(\log n)$, respectively. In Function 3, time complexities of putting map tasks and reduce tasks in respective queues are $O(\mid I \mid (\mid K \mid +1))$ and $O(\mid J \mid (\mid L \mid +1))$, respectively (Function 3, lines 2-5). Similarly, that of lines 7-12 is $O(\mid I \mid\mid K \mid + \mid J \mid\mid L \mid)$. The total time complexity of Function 3 is hence $O(\mid I \mid\mid K \mid + \mid J \mid\mid L \mid)$.

In Algorithm 4, the time complexity of line 16 is $O(\mid K \mid)$. That of removing map tasks and reduce tasks from queues and assigning them to respective slots is $O(\mid I \mid (\log \mid I \mid + \mid K \mid \log \mid K \mid) + \mid J \mid (\log \mid J \mid + \mid L \mid \log \mid L \mid)$ (Algorithm 4, lines 3-15 and also lines 17-29). Therefore, the total time complexity for Algorithm 4 is the latter since it is the dominant one.

Function 4 is the main body of our algorithm, which calls ESTIMATE, SETUPQ and ASSIGN functions. Thus, the time complexity for TGSAVE is the sum of each one of them that is equal to the time complexity of Algorithm 4 since it has the dominant time complexity. The space complexity of the algorithm is $O(\mid I \mid\mid K \mid + \mid J \mid\mid L \mid)$ for the arrays and heaps.

6 Experiments

This section evaluates the performance of our proposed algorithm, TGSAVE. We perform many experiments to analyze our proposed algorithm to show the effect of numbers of map and reduce tasks, size of input data and function of applications and heterogeneity level of environments. First, we describe benchmarks and workloads used for analyzing the performance of algorithms, and then we compare the performance of TGSAVE with that of EMRSA [23] and optimal solution.

6.1 Benchmarks and Workloads

For analyzing the performance of algorithms, we use eleven benchmarks. Since we want to compare TGSAVE with EMRSA, we select Tera Sort, K-Means, and Page-Rank because they were used for evaluating EMRSA. Mashayekhy et al. in [23] classified configurations of Tera Sort into two class of large and small scale. Since small-scale experiments are not sufficient to accurately evaluate the effectiveness

of scheduling algorithm for Big Data applications, we only use large scale configurations. We name the workload related to the number of map tasks and the number of reduce tasks. For instance, workload 128M-256R has 128 map tasks and 256 reduce tasks. These experiments show the effect of numbers of map and reduce tasks on the performance of algorithms. The other ten benchmarks that we use for evaluating algorithms are nine PUMA benchmarks presented by Ahmad et al. [4] plus Page-Rank. We call the set of selected benchmarks PUMA+. We select PUMA+ benchmark suite because it represents a broad range of MapReduce applications demonstrating application characteristics with high and low computation as well as high and low shuffle volumes. Histogram-Movies and Classification are two benchmarks with no reduce tasks. Reduce-less benchmarks are used for evaluating the third phase of our algorithm (Algorithm 4). For these two benchmarks, there is only one choice for the deadline of map phase as there are no reduce tasks and we can assign entire given deadline to the map phase deadline. Name of each workload is the name of the benchmarks since we profile only one configuration for each of them related to input data for the benchmarks. Name and the configuration of each selected PUMA+ workload are shown in Table 1 The benchmarks are described below:

- Word-Count counts the occurrences of each word in the input data. Input data is a large document or a collection of documents. The user defines the term word.
- Sequence-Count is similar to Word-Count. Instead of counting a word, Sequence-Count counts all unique sets of three consecutive words in the input documents.
- Self-join generates association among k+1 fields given the set of k-field associations.
- Inverted-Index generates word to document indexing while the input is a list of documents. Ranked-Inverted-Index takes a list of words and their frequencies per document as input and generates lists of documents that contain the given words. Documents in the output lists are ranked based on the frequency.
- Histogram-Movies is a generic tool used in many data analyses. It generates a histogram of the input data.
- Classification classifies movies into numbers of pre-determined clusters. The input data for both Histogram-Movies and Classification is the same, and it includes key-value pairs where the key is movie ID and value is a list of raters ID and their rating.
- Adjacency-List generates adjacency list and reverses adjacency list of nodes for a given graph.
- Tera Sort samples the input data and uses MapReduce to sort the data into a total order.
- K-means clusters data into k clusters.
- Page-Rank ranks website for use in search engines.

We use these benchmarks to evaluate the performance of algorithms for different applications and different datasets.

6.2 Experimental Setup

For profiling selected PUMA+ workloads [4] and therefore measuring energy consumption and processing time of each task while executing on different machines, we deploy a cluster of nine machines with the total of 40 cores. Configurations of all nine machines are shown in Table 2. The CPU used in all machines is Intel Xeon E5. The deployed cluster is a cluster of Hadoop version 1.2.1. We set one map, and one reduce slot to each core. We set Slowstart parameter of Hadoop to 1, to avoid overlapping of execution of map and reduce tasks to accurately measure the energy consumption of map tasks and reduce tasks separately. Then, we ran PUMA+ workloads on

Table 1 PUMA+ Workloads

Workloads	Number of Tasks	
	Map	Reduce
Word-Count	100	16
Sequence-Count	60	80
Self-Join	135	90
Ranked-Inverted-Index	130	90
Inverted-Index	100	16
Histogram-Movies	240	0
Classification	120	0
Adjacency-List	70	80
K-Means	120	70
Page-Rank	110	70

Table 2 Machines configurations

Machines Configurations	Number Of Cores	RAM (GB)
1	1	3.7
2	2	7.5
3	4	15
4	2	13
5	2	1.8
6	4	3.6
7	8	7.2
8	16	14.4
9	1	1.7

each node of deployed Hadoop cluster for measuring processing time and energy consumption of each task on each node. We use Hadoop logs for measuring processing time of each map and reduce task. Moreover, we use built-in power meter of servers to measure the energy consumption of each map and reduce task. We subtract energy consumption of idle state of machines from measured values to get the exact energy consumption for each map and reduce tasks while executing on different machines.

For Tera Sort workloads, we did not run and profile them. Instead, we used the same profiled data that Mashayekhy et al. [23] used for evaluating EMRSA. They used a cluster of four machines with the total of 80 GB memory and 64 processors. Every node has 16 2.4 GHz Intel processor. Two of them have 24 GB memory, and the other two have 16GB memory.

In our experiments, we first measured the time and energy consumption of tasks on real machines as described above. Then, we analyzed the performance of TGSAVE, EMRSA and optimal solution for minimizing the total energy consumption for all of the prepared workloads through simulations based on these measurements. We implemented EMRSA as Mashayekhy et al. described it in [23]. For getting the optimal result, we solve the ILP model described in Section 4. As solving ILP models may take hours and even days, we ran the ILP solver until the results of ILP reached to below 0.5% of the LP relaxation of the same ILP model. Note that such LP relaxation model gives a lower bound of the solution of the ILP model. The gap between our ILP solution and LP lower bound is only a fraction of a percent, which provides a very close approximation of the optimal results.

We perform two classes of experiments for each workload: relaxed deadline and tight deadline experiments. We define the deadlines smaller than three times the minimum makespan as tight, and on the other hand, deadlines bigger than five times the minimum makespan are considered relaxed deadlines. For relaxed deadline experiments, we select 1500 (2500 for k-means) seconds as given deadline, and for tight deadline experiments, we choose the maximum of the smallest satisfiable deadline of EMRSA and that of TGSAVE as shown in Tables 3 and 4 for Tera Sort large-scale and PUMA+ workloads, respectively. The second column of the tables reports the corresponding value relative to the minimum makespan.

We perform relaxed and tight deadline experiments to compare the performance of TGSAVE and EMRSA in different situations. We also perform experiments to evaluate the tightest satisfiable deadline for each algorithm. We ran both TGSAVE and EMRSA several times in a binary search manner for each workload to find the tightest deadline that each algorithm can meet. We also measure the run time of TGSAVE, EMRSA, and the Optimal on an Intel Core i7 2740QM processor with 8 GB memory. Since run time of Optimal can reach to hours while at a large part of this time the objective function does not noticeably improve, we mark the ILP finished as soon as its objective function reaches to 0.5% proximity of its LP relaxation; note that since the solution to the LP relaxation of the ILP problem represents a lower bound of the original ILP problem, this ensures 0.5% proximity to the ILP optimal solution even in the worst case.

Table 3 Tight deadline for tera sort large-scale workloads

Workloads	Tight Deadline (s)	% of makespan
64M-128R	181	146
128M-64R	164	122
128M-128R	161	130
128M-256R	196	114
128M-512R	257	112
256M-128R	180	136
256M-256R	205	145
256M-512R	291	128
512M-128R	243	175
512M-256R	221	173
512M-512R	329	170

Table 4 Tight deadline for puma+ workloads

Workloads	Tight Deadline (s)	% of makespan
Word-Count	564	236
Sequence-Count	693	274
Self-Join	634	244
Ranked-Inverted-Index	634	238
Inverted-Index	760	311
Histogram-Movies	160	186
Classification	368	396
Adjacency-List	545	142
K-Means	1806	184
Page-Rank	1039	169

6.3 Analysis of Results

We analyzed different aspects of our work, including comparing the performance of TGSAVE with EMRSA and the Optimal on Tera Sort and PUMA+ workloads, analysis of the characteristics of the workloads themselves to determine the root cause of their different performances under above algorithms, and analysis of our proposed algorithm performance when the job size increases.

Tera Sort We perform several experiments on profiled datasets to compare the performance of TGSAVE, EMRSA, and Optimal. Figure 2 presents the total energy consumption of relaxed (a) and tight (b) deadline experiments on Tera Sort workloads obtained by EMRSA, TGSAVE and Optimal. The results show that regarding energy reduction, TGSAVE performs only marginally better than EMRSA for every Tera Sort workload. For these two class of experiments on Tera Sort Benchmarks, TGSAVE achieves energy consumption for jobs up to 4.78% less than EMRSA solutions with an average of 2.66%. Note that in terms of energy reduction, even Optimal is not much better either for this set of benchmarks—see Analysis of Workloads below to find out the reason. Concerning the tightest satisfiable deadline, however, TGSAVE performs significantly better than EMRSA as discussed below.

The time complexity of finding the optimal solution is exponential. This fact makes Optimal impractical because scheduling may take even longer than the job itself. Nevertheless, TGSAVE with polynomial time complexity finds a schedule close to Optimal. For evaluating the difference between TGSAVE and optimal solution, we define Distance to Optimal, which determines the potential of improving TGSAVE energy consumption to reach energy consumption of the optimal solution. Comparing results for TGSAVE and Optimal shows that TGSAVE performance can be enhanced only 2.96% and 2.51% on average for relaxed and tight deadline experiments, respectively. The worst-case Tera Sort workload for the relaxed deadline is the one with 256 map tasks and 128 reduce tasks, in which the result of TGSAVE is 4.30% far away from the result of Optimal. Likewise, Tera Sort large-scale workload with 128 map tasks and 64 reduce tasks is worst-case for tight deadline experiments, where the Distance to Optimal is 4.35%.

Both TGSAVE and EMRSA have a lower bound for given deadline that they can meet. We called it Tightest Satisfiable Deadline (TSD). Obviously, it is better to have a smaller lower bound, which means the algorithm can schedule jobs under harder conditions. Figure 3a presents the TSD of TGSAVE and EMRSA alongside the energy-oblivious minimum makespan for each Tera Sort workload. The results show that for almost every workload TGSAVE can satisfy deadlines tighter than EMRSA. For these experiments, TGSAVE can satisfy deadlines up to 19.89%, and 8.03% on average, smaller than smallest given deadline for which EMRSA can find a feasible schedule while satisfying SLA. In addition, TGSAVE can meet deadlines 19.31% bigger than the minimum makespan on average. Note that, the primary goal of TGSAVE is to minimize energy consumption.

The results presented in Fig. 3b show that TGSAVE performs scheduling only marginally faster than EMRSA as expected by considering the time complexities. Both TGSAVE and EMRSA, are much faster than Optimal. Since time complexity of Optimal is exponential, it quickly becomes impractical when the number of map and reduce tasks increases. Therefore, it makes sense to use a scheduler with better time complexity.

PUMA+ Benchmarks Figure 4a presents total energy consumption obtained by the scheduling of TGSAVE, EMRSA, and Optimal for relaxed deadline experiments on PUMA+ workloads. The results show that TGSAVE improvement over EMRSA is at least

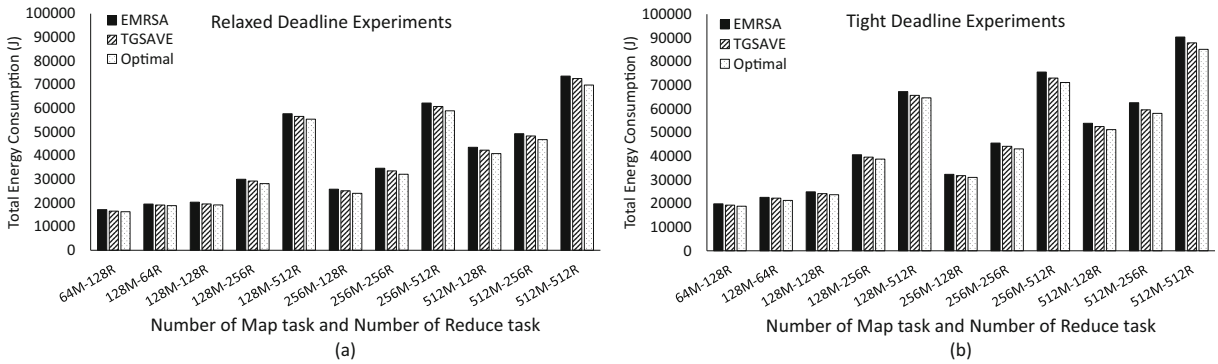

Fig. 2 Total energy consumption of Tera Sort workloads obtained by EMRSA, TGSAVE and Optimal: **a** Results of relaxed deadline experiments; **b** Results of tight deadline experiments

16.68% and at most 35.37% with an average of 25.25%. The improvement for Histogram-Movies and Classification are 21.04% and 16.68% respectively. Figure 4b shows TGSAVE also perform scheduling better than EMRSA for PUMA+ workloads in tight deadline experiments. The improvement for the relaxed deadline experiments is at least 11.59% and at most 30.24% with an average of 19.92%. The Improvement for Histogram-Movies and Classification is 18.94% and 11.59%, which means regardless of choosing map phase deadline, TGSAVE task assignment strategy is better. The results show that in both tight and relaxed deadline experiments classification as a reduce-less benchmark has the least improvement, which means TGSAVE split deadline better into the deadline for map and reduce phase. Note that we count Histogram-Movies and Classification in overall results too.

The Distance to Optimal is at most 1.91% and 4.61% with an average of 0.52% and 1.97% for relaxed and tight deadline experiments, respectively. The results show that scheduling of TGSAVE for five benchmarks perfectly matches the optimal solution in relaxed deadline experiments and the Distance to Optimal is zero percent, while there is only one perfectly match in the tight deadline experiments. Therefore, we can deduce that as much as the deadline is more relaxed, TGSAVE can find an optimal or near optimal solution. The Distance to Optimal for Reduce-less benchmarks is 0.00% and 0.57% on average, respectively for relaxed and tight deadline experiments. Thus, the performance of the third phase of TGSAVE is very close to the optimal assignment.

The TSD of TGSAVE is up to 73.94% smaller than that of EMRSA with an average of 48.68% for PUMA+ workloads, as shown in Fig. 5a. On average, the tightest satisfiable deadline of TGSAVE is only 10.84% far away from the makespan. For Histogram-Movies and Classification workloads, the TSD of EMRSA is 160 and 368 seconds respectively, whereas

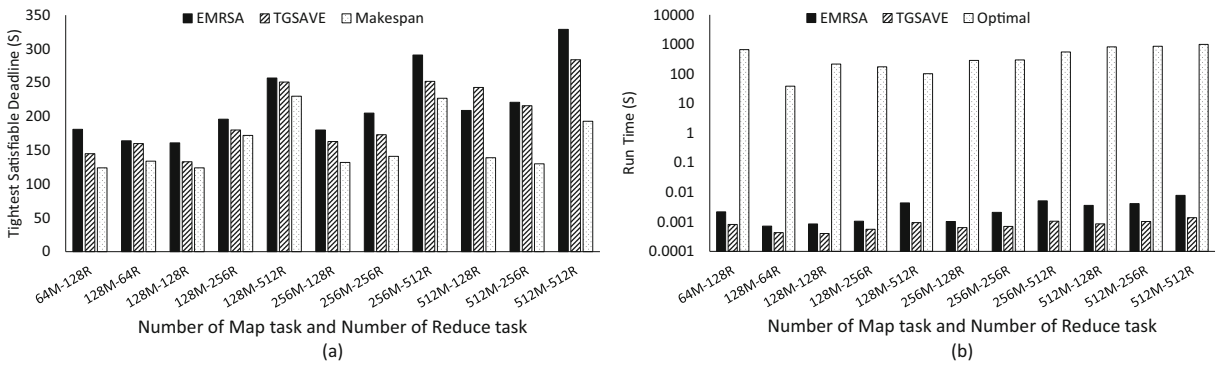

Fig. 3 The performance of TGSAVE and EMRSA for Tera Sort workloads: **a** TSD of TGSAVE and EMRSA alongside the energy-oblivious minimum makespan **b** Run time of TGSAVE, EMRSA, and Optimal

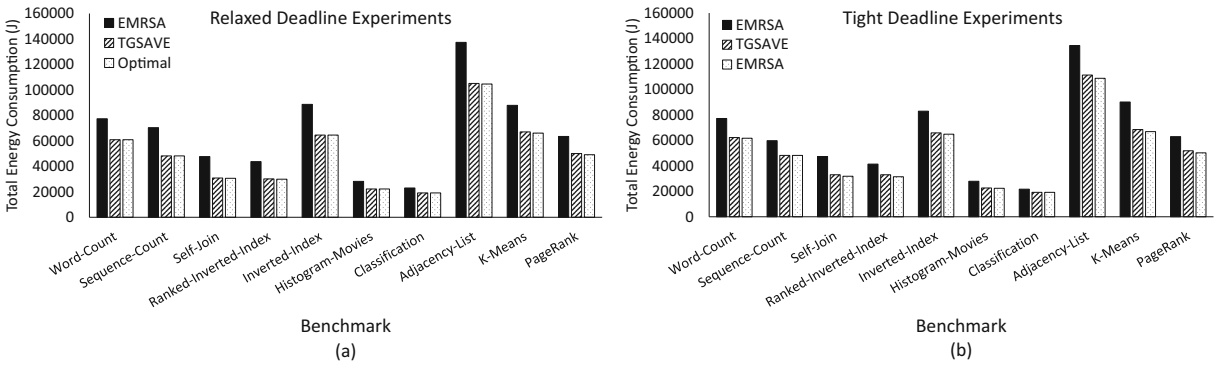

Fig. 4 Total energy consumption of PUMA+ workloads obtained by EMRSA, TGSAVE, and Optimal: **a** Results of relaxed deadline experiments; **b** Results of tight deadline experiments

that numbers for TGSAVE are 88 and 96 seconds while makespan of these workloads are 86 and 93 seconds, respectively. The results show that TGSAVE assigns tasks more efficiently than EMRSA. Hence, TGSAVE is capable of finding solutions in harder conditions. Figure 5b presents run time of EMRSA, TGSAVE, and Optimal. TGSAVE and EMRSA perform in almost the same order of time, and much faster than Optimal. Since the run-time of both algorithms (below 0.01s) is order of magnitude smaller than the run-time of the MapReduce job itself (in order of minutes), the run-time of algorithms is negligible.

Analysis of Workloads Our analysis shows that for every workload TGSAVE minimizes energy more than EMRSA and is closer to the optimal solution while the time complexity of both algorithms is polynomial in the number of map slots, reduce slots, map tasks and reduce tasks. To understand why there is not much difference between the performance of TGSAVE and EMRSA for Tera Sort benchmarks, we designed some more experiments to analyze characteristics of workloads. Tables 5 and 6 depict the energy consumption variance and processing time variance for each workload. The variance of each task is the variance of energy consumption (time) of each task on every slot. The reported number of columns is the average of the variances of every task for each workload. Higher variation in energy and time of workloads means a higher level of heterogeneity in the experiments environments. The difference between solutions of TGSAVE and EMRSA becomes more significant when heterogeneity level of workloads is higher. Since only two different configurations are used for profiling Tera Sort benchmark while nine are used for profiling PUMA+ benchmark, the level of heterogeneity for Tera Sort workloads is much smaller than that of PUMA+ workloads. Therefore, TGSAVE significant improvements over EMRSA are more obvious in our PUMA+ workloads. This not only justifies

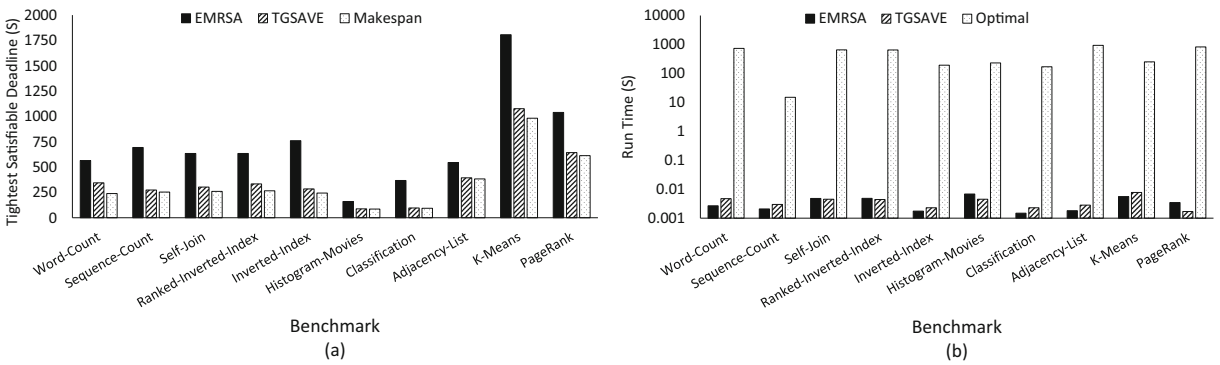

Fig. 5 The performance of TGSAVE and EMRSA for PUMA+ benchmarks workloads: **a** TSD of TGSAVE and EMRSA alongside the energy-oblivious minimum makespan **b** Run time of TGSAVE, EMRSA, and Optimal

Table 5 Variance of energy and time for tera sort workloads

Workloads	Variance	
	Energy	Time
64M-128R	129	73
128M-64R	115	73
128M-128R	184	64
128M-256R	270	73
128M-512R	99	73
256M-128R	125	73
256M-256R	179	73
256M-512R	144	73
512M-128R	135	73
512M-256R	102	72
512M-512R	144	73

the above difference in TGSAVE performance across benchmarks but also demonstrates its significance and superiority in heterogeneous environments which is its main target area of usage.

Analysis of Map Phase Deadline Estimation Choosing a deadline for map phase is the most important part of the first phase of our proposed algorithm. We performed several experiments on every workload. Instead of assigning map phase deadline automatically, we assigned a value manually to it and then ran the rest of the algorithm. We tried a range of values between zero and the given deadline for every workload. The results showed that the choice of TGSAVE for map phase deadline is very close to the best choice: by choosing the absolute best choice for map phase deadline and running the rest of our proposed algorithm as before, only 0.30% improvement is achieved on average for all 36 workloads. The worst case is observed for Ranked-Inverted-Index from PUMA+ tight deadline workloads in which the objective function improvement between the best choice for map phase deadline and TGSAVE choice is only 1.43%: in this case, the given deadline is 634 seconds, and the range of map phase deadlines that TGSAVE can satisfy is from 170 to 498 seconds; TGSAVE chooses 373 seconds for map phase deadline while the best choice is 280 seconds, resulting in 32957 and 32485 joules of energy for original TGSAVE vs. best-map-deadline cases respectively. Since the difference is too small, adding more effort to better estimate the map phase deadline is not worth it.

6.4 Discussion

We used the same Hadoop MapReduce model used in [23], which ignores some challenging issues in MapReduce execution, for a fair comparison of the algorithms.

A MapReduce job is divided to splitting the input data over the cluster of machines, executing the map script on the input data, (optionally) combining and sorting the intermediate data on each machine, shuffling the intermediate data, and reducing the intermediate data [34]. The above model takes only the map, and the reduce phases into account to schedule a MapReduce job and does this scheduling statically as opposed to more recent dynamic scheduling for Hadoop clusters. Moreover, speculative execution of map and reduce tasks comes into play when processing of a task on a machine takes unexpectedly more time than expected [34]. Finally, there are some real-time transient system noises such as data skew and network congestion. These are all known limitations of the above model that needs to be addressed by further research. The purpose and thrust of this paper are to enhance the state of the art in the energy-aware static scheduling of MapReduce jobs in heterogeneous environments as in [23]. Other statically-addressable issues can be readily added to the above model, but dynamic aspects due to runtime anomalies, such as speculative execution, as well as data-dependent

Table 6 Variance of energy and time for PUMA+ workloads

Workloads	Variance	
	Energy	Time
Word-Count	6681	6393
Sequence-Count	9632	7805
Self-Join	1609	7743
Ranked-Inverted-Index	1562	7912
Inverted-Index	13513	9159
Histogram-Movies	250	1115
Classification	953	6500
Adjacency-List	14207	5409
K-Means	5187	15952
Page-Rank	4843	9954

variations, such as shuffling time, need other attacks. In either case, our algorithm and its major points can be the basis for future extensions, and our experimental results can help the research community in understanding the nature of workloads and their behaviors in the energy-aware scheduling of MapReduce jobs. These extensions are indeed part of our future research directions.

7 Conclusion

In this paper, we tackled the problem of scheduling of a single MapReduce job in order to minimize the energy consumption while satisfying service level agreement in a heterogeneous cluster. Minimizing energy consumption of each job is important to reduce energy-related operational expenses as well as the carbon footprint of data centers hosting them. Moreover, one cannot usually sacrifice processing time to save energy. In comparison to EMRSA, our solution not only saves much more energy but also works under heavily tighter deadlines. In this work, we first presented an ILP model for this problem. Then, We proposed a task-based greedy scheduling algorithm, TGSAVE. Since TGSAVE is a greedy algorithm, it schedules jobs very fast. We performed several experiments for eleven different benchmarks exhibiting application characteristics with high and low computation as well as high and low shuffle volumes. The results show that total energy consumption of TGSAVE is below 5% far from the optimal solution and is much better than EMRSA by up to 35% with an average of approximately 20%. In addition, TGSAVE can meet deadlines near the energy-oblivious minimum makespan and find a solution for up to 74% (45% on average) tighter deadlines than the tightest deadline that EMRSA can find a feasible schedule for.

Our plan for future work has three directions. One is to relax the problem and let reduce tasks start before completion of all of the map tasks. Another one is to design and implement a time adaptive multi-job scheduling algorithm in heterogeneous environments with the objective to minimize the energy consumption while meeting individual deadlines of jobs. A final important point to mention is that we did not consider the issues of initial data placement, as well as intermediate data, shuffle phase in this paper, as [23] did not do either. It is definitely important to consider these stages as well, and we are already doing this [25], but since our contribution here is a more effective algorithm for the same problem presented in [23], we used the same problem definition intact. Providing efficient algorithms for the above extended case is thus part of our current and future work.

Acknowledgements The authors would like to thank Seyed Morteza Nabavinejad for his helpful advice and helping us in profiling workloads. We would also like to cordially thank Lena Mashayekhy for kindly providing us with the profiled data of Tera Sort benchmark workloads they used in their experiments.

References

1. Annual Energy Review: Tech. rep. (2012). http://www.eia.gov/totalenergy/data/annual/pdf/aer.pdf (2011)
2. Adaptive Computing, I.: TORQUE Resource Manager. http://www.adaptivecomputing.com/products/open-source/torque/
3. Ahmad, F., Chakradhar, S., Raghunathan, A., Vijaykumar, T.N.: Tarazu : Optimizing MapReduce On Heterogeneous Clusters. In: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems APLOS 40, pp. 61–74 (2012)
4. Ahmad, F., Lee, S., Thottethodi, M., Vijaykumar, T.N.: PUMA: Purdue MapReduce Benchmarks Suite (2012)
5. Anjos, J.C.S., Carrera, I., Kolberg, W., Tibola, A.L., Arantes, L.B., Geyer, C.R.: MRA++: scheduling and data placement on MapReduce for heterogeneous environments. Futur Gener Comput Syst **42**, 22–35 (2015)
6. Apache: Capacity Schedular for Hadoop. https://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html
7. Apache: Hadoop. Hadoop.Apache.org (2016)
8. Apache: Hadoop Fair Scheduler. hadoop.apache.org/docs/r1.2.1/fair_scheduler.html (2016)
9. Apache: HOD Schedular. https://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html (2016)
10. Bryk, P., Malawski, M., Juve, G., Deelman, E.: Storage-aware algorithms for scheduling of workflow ensembles in clouds. Journal of Grid Computing **14**(2), 359–378 (2016)
11. Cho, B., Rahman, M., Chajed, T., Gupta, I.: Natjam: eviction policies for supporting priorities and deadlines in mapreduce clusters (2013)
12. Cisco: Cisco Global Cloud Index : Forecast and Methodology , pp. 2014–2019. Tech. rep. (2014)
13. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM **51**, 107–113 (2008)
14. Ebrahimirad, V., Goudarzi, M., Rajabi, A.: Energy-aware scheduling for precedence-constrained parallel virtual machines in virtualized data centers. Journal of Grid Computing **13**(2), 233–253 (2015)
15. Fredman, M., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. J. Assoc. Comput. Mach. **34**, 596–615 (1987)

16. Guo, Z., Fox, G.: Improving MapReduce performance in heterogeneous network environments and resource utilization. In: Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, pp. 714–716 (2012)
17. Kansal, N.J., Chana, I.: Energy-aware virtual machine migration for cloud computing-a firefly optimization approach. Journal of Grid Computing **14**(2), 327–345 (2016)
18. Kim, H., Ahn, J.H., Kim, J.: Exploiting replicated cache blocks to reduce L2 cache leakage in CMPs. IEEE Transactions on Very Large Scale Integration (VLSI) Systems (10), 1863–1877 (2013)
19. Krish, K., Anwar, A., Butt, A.R.: [phi]Sched: A Heterogeneity-Aware Hadoop Workflow Scheduler. In: 2014 IEEE 22nd International Symposium on Modelling, Analysis &, Simulation of Computer and Telecommunication Systems, pp. 255–264 (2014)
20. Lang, W., Patel, J.M.: Energy management for MapReduce clusters. Proceedings of the VLDB Endowment **3**, 129–139 (2010)
21. Leverich, J., Kozyrakis, C.: On the energy (in)efficiency of Hadoop clusters. ACM SIGOPS Operating Systems Review **44**, 61–65 (2010)
22. Marszałkowski, J.M., Drozdowski, M., Marszałkowski, J.: Time and energy performance of parallel systems with hierarchical memory. Journal of Grid Computing **14**(1), 153–170 (2016)
23. Mashayekhy, L., Movahed Nejad, M., Grosu, D., Zhang, Q., Shi, W.: Energy-aware Scheduling of MapReduce Jobs for Big Data Applications. IEEE Transactions on Parallel and Distributed Systems **26**, 2720–2733 (2015)
24. Meisner, D., Gold, B.T., Wenisch, T.F.: PowerNap. ACM SIGARCH Computer Architecture News **37**, 205 (2009)
25. Nabavinejad, S.M., Goudarzi, M., Abedi, S.: MapReduce Service Provisioning for Frequent Jobs on Green Clouds Considering Data Transfers. Technical Report, Computer Engineering Department Sharif University of Technology (2016)
26. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Energy efficiency across programming languages: how do energy, time, and memory relate? In: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, pp. 256–267. ACM (2017)
27. Powell, M.D., Yang, S.H., Falsafi, B., Roy, K., Vijaykumar, T.N.: An Energy-Efficient High-Performance Deep-Submicron instruction cache. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1–13 (2001)
28. Rasooli, A., Down, D.G.: Guidelines for selecting hadoop schedulers based on system heterogeneity. Journal of grid computing **12**(3), 499–519 (2014)
29. Sueur, E.L., Heiser, G.: Dynamic voltage and frequency scaling: The laws of diminishing returns. In: Proceedings of the 2010 international conference on Power aware computing and systems, pp. 1–8 (2010)
30. Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S.U., Li, K.: An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment. Journal of Grid Computing **14**(1), 55–74 (2016)
31. Tavarageri, S., Sadayappan, P.: A compiler analysis to determine useful cache size for energy efficiency. In: 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, pp. 923–930 (2013)
32. Tian, C., Zhou, H., He, Y., Zha, L.: A dynamic mapreduce scheduler for heterogeneous workloads 2009 Eighth International Conference on Grid and Cooperative Computing, pp. 218–224 (2009)
33. Wang, Y., Lu, W., Lou, R., Wei, B.: Improving mapreduce performance with partial speculative execution. Journal of Grid Computing **13**(4), 587–604 (2015)
34. White, T.: Hadoop: The Definitive Guide, O'Reilly Media, Inc (2012)
35. Wolf, J., Rajan, D., Hildrum, K., Khandekar, R., Kumar, V., Parekh, S., Wu, K.L., Balmin, A.: FLEX: a slot allocation scheduling optimizer for MapReduce workloads. In: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 1–20 (2010)
36. Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., Qin, X.: Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. Parallel & Distributed Processing. In: 2010 IEEE International Symposium on Workshops and Phd Forum (IPDPSW) 9, pp. 29–42 (2010)
37. Yan, F., Cherkasova, L., Zhang, Z., Smirni, E.: DyScale: a mapreduce job scheduler for heterogeneous multicore processors (2015)
38. Yang, S.J., Chen, Y.R.: Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds. J. Netw. Comput. Appl. **57**, 61–70 (2015)
39. Yigitbasi, N., Datta, K., Jain, N., Willke, T.: Energy efficient scheduling of MapReduce workloads on heterogeneous clusters. In: Proceedings of the 2nd International Workshop - GCM '11 on Green Computing Middleware, pp. 1–6 (2011)
40. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving mapreduce performance in heterogeneous environments. Proceedings of the USENIX OSDI, pp. 8 (2008)
41. Zhang, Q., Zhani, M.F., Boutaba, R., Hellerstein, J.L.: Dynamic heterogeneity-aware resource provisioning in the cloud. In: Distributed Computing Systems (ICDCS), 2013 IEEE 33Rd International Conference on, pp. 510–519 (2013)