Energy-Aware Scheduling of MapReduce Jobs for Big Data Applications

Lena Mashayekhy, *Student Member, IEEE*, Mahyar Movahed Nejad, *Student Member, IEEE*,
Daniel Grosu, *Senior Member, IEEE*, Quan Zhang, *Student Member, IEEE*, and
Weisong Shi, *Senior Member, IEEE*

Abstract—The majority of large-scale data intensive applications executed by data centers are based on MapReduce or its open-source implementation, Hadoop. Such applications are executed on large clusters requiring large amounts of energy, making the energy costs a considerable fraction of the data center's overall costs. Therefore minimizing the energy consumption when executing each MapReduce job is a critical concern for data centers. In this paper, we propose a framework for improving the energy efficiency of MapReduce applications, while satisfying the service level agreement (SLA). We first model the problem of energy-aware scheduling of a single MapReduce job as an Integer Program. We then propose two heuristic algorithms, called energy-aware MapReduce scheduling algorithms (EMRSA-I and EMRSA-II), that find the assignments of map and reduce tasks to the machine slots in order to minimize the energy consumed when executing the application. We perform extensive experiments on a Hadoop cluster to determine the energy consumption and execution time for several workloads from the HiBench benchmark suite including TeraSort, PageRank, and K-means clustering, and then use this data in an extensive simulation study to analyze the performance of the proposed algorithms. The results show that EMRSA-I and EMRSA-II are able to find near optimal job schedules consuming approximately 40 percent less energy on average than the schedules obtained by a common practice scheduler that minimizes the makespan.

Index Terms—MapReduce, big data, minimizing energy consumption, scheduling

1 Introduction

Several businesses and organizations are faced with an ever-growing need for analyzing the unprecedented amounts of available data. Such need challenges existing methods, and requires novel approaches and technologies in order to cope with the complexities of big data processing. One of the major challenges of processing data intensive applications is minimizing their energy costs. Electricity used in US data centers in 2010 accounted for about 2 percent of total electricity used nationwide [1]. In addition, the energy consumed by the data centers is growing at over 15 percent annually, and the energy costs make up about 42 percent of the data centers' operating costs [2]. Considering that server costs are consistently falling, it should be no surprise that in the near future a big percentage of the data centers' costs will be energy costs. Therefore, it is critical for the data centers to minimize their energy consumption when offering services to customers.

Big data applications run on large clusters within data centers, where their energy costs make energy efficiency of executing such applications a critical concern. MapReduce [3] and its open-source implementation, Hadoop [4], have emerged as the leading computing platforms for big data analytics. For scheduling multiple MapReduce jobs, Hadoop originally employed a FIFO scheduler. To overcome the issues with the waiting time in FIFO, Hadoop then employed the Fair Scheduler [5]. These two schedulers, however, do not consider improving the energy efficiency when executing MapReduce applications. Improving energy efficiency of MapReduce applications leads to a significant reduction of the overall cost of data centers. In this paper, we design MapReduce scheduling algorithms that improve the energy efficiency of running each individual application, while satisfying the service level agreement (SLA). Our proposed scheduling algorithms can be easily incorporated and deployed within the existing Hadoop systems.

In most of the cases, processing big data involves running production jobs periodically. For example, Facebook processes terabytes of data for spam detection daily. Such production jobs allow data centers to use job profiling techniques in order to get information about the resource consumption for each job. Job profiling extracts critical performance characteristics of map and reduce tasks for each underlying application. Data centers can use the knowledge of extracted job profiles to pre-compute new estimates of jobs' map and reduce stage durations, and then construct an optimized schedule for future executions. Furthermore, the energy consumption of each task on a machine can be profiled using automatic powermeter tools such as PDU Power Strip [6], which is currently a standard practice in data centers. Many researchers studied different profiling techniques [7], [8], and several MapReduce scheduling studies rely on such techniques [9], [10]. Our proposed algorithms schedule

- *The authors are with the Department of Computer Science, Wayne State University, Detroit, MI 48202.*
  E-mail: {mlena, mahyar, dgrosu, fd7710, weisong}@wayne.edu.

Manuscript received 20 June 2014; revised 5 Sept. 2014; accepted 11 Sept. 2014. Date of publication 0 . 0000; date of current version 0 . 0000.
Recommended for acceptance by X. Gu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2358556

MapReduce production jobs having as the primary objective the minimization of energy consumption.

Most of the existing research on MapReduce scheduling focused on improving the makespan (i.e., minimizing the time between the arrival and the completion time of an application) of the MapReduce job's execution (e.g., [11], [12], [13], [14]). However, makespan minimization is not necessarily the best strategy for data centers. Data centers are obligated to deliver the services by their specified deadlines, and it is not in their best interests to execute the services as fast as they can in order to minimize the makespan. This strategy fails to incorporate significant optimization opportunities available for data centers to reduce their energy costs. The majority of production MapReduce workloads consists of a large number of jobs that do not require fast execution. By taking into account the energy consumed by the map and reduce tasks when making scheduling decisions, the data centers can utilize their resources efficiently and reduce the energy consumption. Our proposed energy-aware scheduling algorithms capture such opportunities and significantly reduce the MapReduce energy costs, while satisfying the SLA.

1.1 Our Contribution

To the best of our knowledge this is the first study that designs algorithms for *detailed task placement* of a MapReduce job to machines with the primary focus on minimizing the energy consumption. Our proposed algorithms can be incorporated into higher level energy management policies in data centers. We first model the problem of scheduling MapReduce tasks for energy efficiency as an integer program. In the absence of computationally tractable optimal algorithms for solving this problem, we design two heuristic algorithms, called EMRSA-I and EMRSA-II, where EMRSA is an acronym for energy-aware MapReduce scheduling algorithm. EMRSA-I and EMRSA-II provide very fast solutions making them suitable for deployment in real production MapReduce clusters. The time complexity of the proposed algorithms is polynomial in the number of map and reduce slots, the number of map tasks, and the number of reduce tasks. We perform experiments on a Hadoop cluster to determine the energy consumption of several MapReduce benchmark applications such as TeraSort, Page Rank, and K-means clustering. We use this data in an extensive simulation study to characterize the performance of the proposed algorithms. We show that the current practice scheduling methods, such as makespan minimization, produce schedules having energy consumption that is far from optimal. We compare the performance of EMRSA-I and EMRSA-II against the optimal solution for cases in which the optimal solution can be obtained in reasonable amount of time. The results show that EMRSA-I and EMRSA-II are capable of finding close to optimal solutions very fast. Due to the intractability of the problem, when the optimal results are not available, we show that the energy consumption for the schedules obtained by the proposed algorithms is very close to the lower bound solution obtained by the linear programming (LP) relaxation of the integer program.

1.2 Related Work

We summarize the related work from three perspectives: resource allocation and scheduling in data centers and clouds, MapReduce scheduling with different objectives, and energy savings in data centers.

Resource allocation and scheduling in data centers and clouds. Hacker and Mahadik [15] proposed scheduling polices for virtual high performance computing clusters. They presented a resource prediction model for each policy to estimate the resources needed within a cloud, the queue wait time for requests, and the size of the pool of spare resources needed. Palanisamy et al. [16] proposed a new MapReduce cloud service model for production jobs. Their method creates cluster configurations for the jobs using MapReduce profiling and leverages deadline-awareness, allowing the cloud provider to optimize its global resource allocation and reduce the cost of resource provisioning. Ekanayake et al. [17] proposed a programming model and an architecture to enhance MapReduce runtime that supports iterative MapReduce computations efficiently. They showed how their proposed model can be extended to more classes of applications for MapReduce. Tian and Chen [18] proposed a cost function that models the relationship between the amount of input data, Map and Reduce slots, and the complexity of the Reduce function for the MapReduce job. Their proposed cost function can be used to minimize the cost with a time deadline or minimize the time under certain budget. Zhan et al. [19] proposed a cooperative resource provisioning solution using statistical multiplexing to save the server cost. Song et al. [20] proposed a two-tiered on-demand resource allocation mechanism consisting of the local and global resource allocation. In our previous studies [21], [22], [23], we proposed mechanisms for resource provisioning, allocation, and pricing in clouds considering several heterogeneous resources. However, none of the above mentioned studies consider the energy saving objectives.

MapReduce scheduling with different objectives. Zaharia et al. [24] studied the problem of speculative execution in MapReduce. They proposed a simple robust scheduling algorithm, Longest Approximate Time to End (LATE), which uses estimated finish times to speculatively execute the tasks that hurt the response time the most. Sandholm and Lai [25] designed a system for allocating resources in shared data and compute clusters that improves MapReduce job scheduling. Their approach is based on isolating MapReduce clusters in VMs with a continuously adjustable performance. Wang et al. [26] proposed a task scheduling technique for MapReduce that improves the system throughput in job-intensive environments without considering the energy consumption. Ren et al. [27] proposed a job scheduling algorithm to optimize the completion time of small MapReduce jobs. Their approach extends job priorities to guarantee the rapid response for small jobs. Chang et al. [12] proposed various online and offline algorithms for the MapReduce scheduling problem to minimize the overall job completion times. Their algorithms are based on solving a linear program relaxation. Moseley et al. [11] proposed a dynamic program for minimizing the makespan when all MapReduce jobs arrive at the same time. They modeled the problem as a two-stage flow shop problem, and proved that the dynamic program yields a PTAS if there is a fixed number of job-types. Pastorelli et al. [28] proposed a size-based approach to scheduling jobs in Hadoop to guarantee fairness and near-optimal system response times. Their scheduler requires a priori job size information, and thus, it builds such knowledge by estimating the sizes during job execution. Wolf et al. [29] proposed a flexible scheduling allocation scheme, called Flex, to optimize a variety of standard scheduling metrics such as response time and makespan, while ensuring the same minimum job slot guarantees as in the case of Fair scheduler. Sandholm and Lai [30] proposed a dynamic priority parallel task scheduler for Hadoop that prioritizes jobs and users and gives users the tool to optimize and customize their allocations to fit the importance and requirements of their jobs such as deadline and budget. Verma et al. [8] proposed a job scheduler for MapReduce environments that allocates the resources to production jobs. Their method can profile a job that runs routinely and then uses its profile in the designed MapReduce model to estimate the amount of resources required for meeting the deadline. Verma et al. [9] proposed a job scheduler that minimizes the makespan for MapReduce production jobs with no dependencies by utilizing the characteristics and properties of the jobs in a given workload. Nanduri et al. [31] proposed a heuristic scheduling algorithm to maintain a resource balance on a cluster, thereby reducing the overall runtime of the MapReduce jobs. Their job selection and assignment algorithms select the job that is best suitable on a particular node while avoiding node overloads. Ibrahim et al. [32] proposed a scheduling algorithm for map tasks to improve the overall performance of the MapReduce computation. Their approach leads to a higher locality in the execution of map tasks and to a more balanced intermediate data distribution. Kurazumi et al. [33] proposed dynamic processing slot scheduling for I/O intensive MapReduce jobs that use efficiently the CPU resources with low utilization caused by I/O wait related to task execution. However, these studies did not consider energy efficiency as their objectives.

Energy savings in data centers. Kaushik et al. [34] proposed an approach to partition the servers in a Hadoop cluster into hot and cold zones based on their performance, cost, and power characteristics, where hot zone servers are always powered on and cold zone servers are mostly idling. Cardosa et al. [35] proposed a spatio-temporal tradeoff that includes efficient spatial placement of tasks on nodes and temporal placement of nodes with tasks having similar runtimes in order to maximize utilization. Leverich and Kozyrakis [36] proposed a method for energy management of MapReduce jobs by selectively powering down nodes with low utilization. Their method uses a cover set strategy that exploits the replication to keep at least one copy of a data-block. As a result, in low utilization periods some of the nodes that are not in the cover set can be powered down. Chen et al. [37] proposed a method for reducing the energy consumption of MapReduce jobs without relying on replication. Their approach divides the jobs into time-sensitive and less time-sensitive jobs, where the former are assigned to a small pool of dedicated nodes, and the latter can run on the rest of the cluster. Maheshwari et al. [38] proposed an algorithm that dynamically reconfigures clusters by scaling up and down the number of nodes based on the cluster utilization. Lang and Patel [39] proposed a framework for energy management in MapReduce clusters by powering down all nodes in the cluster during a low utilization period. Wirtz and Ge [40] conducted an experimental study on the MapReduce efficiency. They analyzed the effects of changing the number of concurrent worker nodes, and the effects of adjusting the processor frequency based on workloads. Goiri et al. [41] proposed a MapReduce framework for a data center powered by renewable sources of energy such as solar or wind, and by the electrical grid for backups. Their proposed framework schedules jobs to maximize the green energy consumption by delaying many background computations within the jobs' bounded time. Salehi et al. [42] proposed an adaptive energy management policy employing a fuzzy reasoning engine to determine if the resources for a request have to be allocated through switching on resources, preemption, consolidation, or a combination of these. Shen and Wang [43] formulated several stochastic optimization models to investigate the tradeoff between energy footprints and quality of service in cloud computing services. In their models, decisions include workload scheduling and switching servers on/off based on loads. While the above frameworks can be used as data center-level energy minimization strategies, our focus is on minimizing the energy consumption by scheduling jobs, which can be considered as a cluster-level strategy in data centers. In addition, none of the above frameworks and systems exploit the job profiling information when making the decisions for task placement on the nodes to increase the energy efficiency of executing MapReduce jobs. Our proposed algorithms consider the significant energy consumption differences of different task placements on machines, and find an energy efficient assignment of tasks to machines.

1.3 Organization

The rest of the paper is organized as follows. In Section 2, we describe the problem of scheduling MapReduce jobs for energy efficiency. In Section 3, we present our proposed algorithms. In Section 4, we evaluate the algorithms by extensive experiments. In Section 5, we summarize our results and present possible directions for future research.

2 Energy-Aware MapReduce Scheduling Problem

A MapReduce job comprising a specific number of map and reduce tasks is executed on a cluster composed of multiple machines. The job's computation consists of a map phase followed by a reduce phase. In the map phase, each map task is allocated to a map slot on a machine, and processes a portion of the input data producing key-value pairs. In the reduce phase, the key-value pairs with the same key are then processed by a reduce task allocated to a reduce slot. As a result, the reduce phase of the job cannot begin until the map phase ends. At the end, the output of the reduce phase is written back to the distributed file system. In Hadoop, job scheduling is performed by a master node running a job tracker process, which distributes jobs to a number of worker nodes in the cluster. Each worker runs a task tracker process, and it is configured with a fixed number of map and reduce slots. The task tracker periodically sends heartbeats to the job tracker to report the number of free slots and the progress of the running tasks.

We consider a big data application consisting of a set of M map and R reduce tasks that needs to be completed by deadline D. Tasks in each set can be run in parallel, but no reduce task can be started until all map tasks for the application are completed. Let $\mathcal{M}$ and $\mathcal{R}$ be the set of map and reduce tasks of the application, and $\mathcal{A}$ and $\mathcal{B}$ the set of slots on heterogeneous machines available for executing the map and the reduce tasks, respectively. The number of slots for each machine is decided by the system administrators when the Hadoop cluster is setup and each slot can handle only one map or reduce task at a time. Since we consider a heterogeneous cluster, the execution speed of a task on different slots from different machines may not be the same. Also, the energy required to execute a task on different slots may not be the same. We denote by e_{ij} the difference between energy consumption of slot $j \in \{\mathcal{A}, \mathcal{B}\}$ when executing task $i \in \{\mathcal{M}, \mathcal{R}\}$ and its idle energy consumption. In addition, we denote by p_{ij} the processing times of task $i \in \{\mathcal{M}, \mathcal{R}\}$ when executed on slot $j \in \{\mathcal{A}, \mathcal{B}\}$. We assume that the processing time of the tasks are known. In doing so, we use the knowledge of extracted job profiles to pre-compute the processing time of map and reduce tasks, along with their energy consumption. We define an indicator variable δ_{ti}, $\forall t, i \in \mathcal{M} \cup \mathcal{R}$, characterizing the dependencies of the map and reduce tasks as follows:

$$\delta_{ti} = \begin{cases} 1, & \text{if task } i \text{ should be assigned after task } t, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

We formulate the energy-aware MapReduce scheduling problem as an integer program (called EMRS-IP), as follows:

$$\text{Minimize} \sum_{j\in\mathcal{A}} \sum_{i\in\mathcal{M}} e_{ij} X_{ij} + \sum_{j\in\mathcal{B}} \sum_{i\in\mathcal{R}} \sum_{t\in\mathcal{M}\cup\mathcal{R}} \delta_{ti} e_{ij} Y_{ij}, \tag{2}$$

Subject to :

$$\sum_{j\in\mathcal{A}} X_{ij} = 1, \forall i \in \mathcal{M} \tag{3}$$

$$\sum_{j\in\mathcal{B}} \sum_{t\in\mathcal{M}\cup\mathcal{R}} \delta_{ti} Y_{ij} = 1, \forall i \in \mathcal{R} \tag{4}$$

$$\sum_{i\in\mathcal{M}} p_{ij} X_{ij} + \sum_{i\in\mathcal{R}} \sum_{t\in\mathcal{M}\cup\mathcal{R}} \delta_{ti} p_{ij'} Y_{ij'} \leq D, \quad \forall j \in \mathcal{A}, \forall j' \in \mathcal{B} \tag{5}$$

$$X_{ij} = \{0, 1\}, \forall i \in \mathcal{M}, \forall j \in \mathcal{A} \tag{6}$$

$$Y_{ij} = \{0, 1\}, \forall i \in \mathcal{R}, \forall j \in \mathcal{B}, \tag{7}$$

where the decision variables X_{ij} and Y_{ij} are defined as follows:

$$X_{ij} = \begin{cases} 1, & \text{if map task } i \text{ is assigned to slot } j, \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

$$Y_{ij} = \begin{cases} 1, & \text{if reduce task } i \text{ is assigned to slot } j, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

The objective function is to minimize the energy consumed when executing the MapReduce application considering the dependencies of reduce tasks on the map tasks. Constraints (3) ensure that each map task is assigned to a slot for execution. Constraints (4) ensure that each reduce task is assigned to a slot. Constraints (5) ensure that processing time of the application does not exceed its deadline. Constraints (6) and (7) represent the integrality requirements for the decision variables. The solution to EMRS-IP consists of X and $\hat{Y}$, where $\hat{Y}_{ij} = \sum_{t\in\mathcal{M}\cup\mathcal{R}} \delta_{ti} Y_{ij}$, $i \in \mathcal{R}$, and $j \in \mathcal{B}$.

Note that based on Constraints (5), the scheduler can assign all reduce tasks after finishing all map tasks without exceeding the deadline. This is due to the fact that these constraints can be interpreted as $\max_{\forall j\in\mathcal{A}} \sum_{i\in\mathcal{M}} p_{ij} X_{ij} + \max_{\forall j'\in\mathcal{B}} \sum_{i\in\mathcal{R}} p_{ij'} Y_{ij'} \leq D$. As a result, all reduce tasks can be assigned after time $\max_{\forall j\in\mathcal{A}} \sum_{i\in\mathcal{M}} p_{ij} X_{ij}$. In addition, the scheduler can assign multiple map tasks to a machine, as well as multiple reduce tasks. This is due to the fact that in bigdata applications the number of tasks is greater than the number of machines available in a cluster. The focus of this study is the detailed placement of map and reduce tasks of a job in order to reduce energy consumption. While it is important to consider data placement in an integrated framework for energy savings in data centers, data placement is beyond the scope of this paper.

At the high level the problem we consider may appear as composed of two independent scheduling problems, one for the map tasks and one for the reduce tasks. This would be the case if the deadline for the map phase would be known. But since the deadline for map tasks is not known from the beginning, we cannot just simply divide the problem into two scheduling subproblems and solve them independently. Our proposed algorithms determine the map deadline as the tasks are allocated and schedule the map and reduce tasks to reduce the energy consumption of executing the job.

3 Energy-Aware MapReduce Scheduling Algorithms

We design two heuristic algorithms called EMRSA-I and EMRSA-II for solving the energy-aware MapReduce scheduling problem. Our proposed algorithms, EMRSA-I and EMRSA-II, take the energy efficiency differences of different machines into account and determine a detailed task placement of a MapReduce job into slots while satisfying the user specified deadline. The two algorithms are presented as a single generic algorithm called EMRSA-X, in Algorithm 1.

The design of these algorithms require a metric that characterizes the energy consumption of each machine and induces an order relation among the machines. We define such a metric, called *energy consumption rate* of a slot j. EMRSA-I and EMRSA-II use different energy consumption rate metrics as follows:

Algorithm 1. EMRSA-X

```
1:  Create an empty priority queue $\mathcal{Q}^m$
2:  Create an empty priority queue $\mathcal{Q}^r$
3:  for all $j \in \mathcal{A}$ do
4:      $ecr^m_j = \min_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}$, for EMRSA-I; or
        $ecr^m_j = \frac{\sum_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}$, for EMRSA-II
5:      $\mathcal{Q}^m$.enqueue$(j, ecr^m_j)$
6:  for all $j \in \mathcal{B}$ do
7:      $ecr^r_j = \min_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}$, for EMRSA-I; or
        $ecr^r_j = \frac{\sum_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}}{R}$, for EMRSA-II
8:      $\mathcal{Q}^r$.enqueue$(j, ecr^r_j)$
9:  $D^m \leftarrow \infty$; $D^r \leftarrow \infty$
10: while $\mathcal{Q}^m$ is not empty and $\mathcal{Q}^r$ is not empty do
11:     $j^m = \mathcal{Q}^m$.extractMin()
12:     $j^r = \mathcal{Q}^r$.extractMin()
13:     $f = \frac{\sum_{\forall i \in \mathcal{M}} p_{ij^m}}{\sum_{\forall i \in \mathcal{R}} p_{ij^r}}$
14:     $\mathcal{T}^m$: sorted unassigned map tasks $i \in \mathcal{M}$ based on $p_{ij^m}$
15:     $\mathcal{T}^r$: sorted unassigned reduce tasks $i \in \mathcal{R}$ based on $p_{ij^r}$
16:     if $\mathcal{T}^m = \emptyset$ and $\mathcal{T}^r = \emptyset$ then break
17:     ASSIGN-LARGE()
18:     ASSIGN-SMALL()
19:     if $D^m = \infty$ then
20:         $D^m = D - p^r$
21:         $D^r = p^r$
22: if $\mathcal{T}^m \neq \emptyset$ or $\mathcal{T}^r \neq \emptyset$ then
23:     No feasible schedule
24:     return
25: Output: $X, Y$
```

1) EMRSA-I uses energy consumption rate metrics based on the minimum ratio of energy consumption and processing time of tasks when executed on slot j, as follows:

$$ecr^m_j = \min_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}, \forall j \in \mathcal{A}, \tag{10}$$

$$ecr^r_j = \min_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}, \forall j \in \mathcal{B}, \tag{11}$$

where ecr^m_j and ecr^r_j represent the energy consumption rate of map slot j and reduce slot j, respectively.

2) EMRSA-II uses energy consumption rate metrics based on the average ratio of energy consumption and processing time of tasks when executed on slot j, as follows:

$$ecr^m_j = \frac{\sum_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}, \forall j \in \mathcal{A}, \tag{12}$$

$$ecr^r_j = \frac{\sum_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}}{R}, \forall j \in \mathcal{B}. \tag{13}$$

The ordering induced by these metrics on the set of slots determines the order in which the slots are assigned to tasks, that is, a lower ecr^m_j means that slot j has a higher priority to have a map task assigned to it. Similarly, a lower ecr^r_j means that slot j has a higher priority to have a reduce task assigned to it.

In addition, EMRSA-X uses the ratio of map and reduce processing times, denoted by f, in order to balance the assignment of map and reduce tasks. The ratio f is defined as follows:

$$f = \frac{\sum_{\forall i \in \mathcal{M}} p_{ij^m}}{\sum_{\forall i \in \mathcal{R}} p_{ij^r}}. \tag{14}$$

This ratio is used in the task assignment process in each iteration of EMRSA-X. As we already mentioned, we use job profiling of production jobs to estimate the processing time of map and reduce tasks. This information, extracted from job profiling (i.e., the values of p_{ij^m} and p_{ij^r}) is used by EMRSA-X to compute the ratio f.

A key challenge when designing the algorithms is that the user only specifies the deadline for the job and there is no information on the deadline for completing the map phase. However, since the reduce tasks are dependent on the map tasks, the algorithms have to determine a reasonable deadline for the map tasks with respect to the availability of the map slots in the cluster in order to utilize its resources efficiently. Our proposed algorithms find the assignments of map tasks to the map slots satisfying the determined map deadline, and then find the assignments of reduce tasks to the reduce slots satisfying the deadline D, where all the reduce tasks start after the map deadline.

First, EMRSA-X determines the assignment of large tasks in terms of their processing time, and the map deadline according to such tasks. The reason that EMRSA-X gives priority to large tasks is due to the hard deadline constraint, and the fact that there may not be many choices for large task placement configurations to avoid exceeding the deadline constraint. Then, EMRSA-X tries to close the optimality gap by filling with smaller tasks the leftover time of each slot based on the deadline. This leads to better utilization of each machine in the cluster.

EMRSA-X is given in Algorithm 1. EMRSA-X builds two priority queues $\mathcal{Q}^m$ and $\mathcal{Q}^r$ to keep the order of the map and reduce slots based on their energy consumption rates (lines 1-8). Then, it initializes the deadlines for map tasks, D^m, and reduce tasks, D^r, to infinity. In each iteration of the while loop, the algorithm chooses the slots with the lowest energy consumption rates (i.e., j^m and j^r) from the priority queues, and finds the task placement on the selected slots. For these slots, the ratio of processing time of map tasks to that of the reduce tasks, denoted by f, is calculated (line 13). Then, EMRSA-X sorts the unassigned map and reduce tasks, if there is any, based on their processing time on the selected slots (lines 14-15). Then, it determines the assignments of large tasks based on the metric f by calling ASSIGN-LARGE() (given in Algorithm 2). Then, it finds the assignments of small tasks by calling ASSIGN-SMALL() (given in Algorithm 3) if there is any unallocated processing time on a slot. EMRSA-X assigns a new task to a slot whenever the slot becomes available. At the end of the first iteration, the algorithm sets the map and reduce deadlines based on the allocated tasks (lines 19-21).

We now describe the two procedures, ASSIGN-LARGE() and ASSIGN-SMALL() into more details. ASSIGN-LARGE() is given in Algorithm 2. ASSIGN-LARGE() selects the longest map task i^m and reduce task i^r from the sorted sets $\mathcal{T}^m$

TABLE 1
Example: Map Tasks

		Map tasks					
		Processing time			Energy consumption		
		a_1	a_2	a_3	a_1	a_2	a_3
Tasks	t_1^m	8	4	2	8	12	12
	t_2^m	3	2	1	3	4	3

TABLE 2
Example: Reduce Tasks

		Reduce tasks			
		Processing time		Energy consumption	
		b_1	b_2	b_1	b_2
Tasks	t_1^r	2	3	6	3
	t_2^r	2	2	6	4

and $\mathcal{T}^r$, respectively (lines 1-2). Then it checks the feasibility of allocating map task i^m to slot j^m and reduce task i^r to slot j^r by checking the total processing time of the tasks against the deadline D (line 4). If the assignment of map task i^m and reduce task i^r is feasible, the algorithm continues to select tasks from $\mathcal{T}^m$ and $\mathcal{T}^r$, and updates the variables accordingly (lines 5-23). To keep the assignments of the tasks in alignment with the ratio of processing time f, the procedure balances the assignment. In doing so, if $f > 1$ (i.e., the load of processing time of map tasks is greater than that of reduce tasks) and the ratio of the current assignment is less than f, then the algorithm assigns more map tasks to balance the allocated processing time close to f (lines 15-20). If the ratio of the current assignment is greater than f, the procedure assigns more reduce tasks to balance the allocated processing time (line 22).

Algorithm 2. ASSIGN-LARGE()

```
 1: $i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
 2: $i^r = \text{argmax}_{t \in \mathcal{T}^r} p_{tj^r}$
 3: $p^m = 0; p^r = 0$
 4: if $p_{i^m j^m} + p_{i^r j^r} \le D$ and $p_{i^m j^m} \le D^m$ and $p_{i^r j^r} \le D^r$ then
 5:   $\mathcal{T}^m = \mathcal{T}^m \setminus \{i^m\}$
 6:   $\mathcal{T}^r = \mathcal{T}^r \setminus \{i^r\}$
 7:   $p^m = p_{i^m j^m}$
 8:   $p^r = p_{i^r j^r}$
 9:   $X_{i^m j^m} = 1$
10:   $Y_{i^r j^r} = 1$
11:   do
12:     $i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
13:     $i^r = \text{argmax}_{t \in \mathcal{T}^r} p_{tj^r}$
14:     if $f > 1$ then
15:       while $\frac{p^m + p_{i^m j^m}}{p^r} < f$ and $p^m + p^r + p_{i^m j^m} \le D$
          and $p^m + p_{i^m j^m} \le D^m$ and $\mathcal{T}^m \neq \emptyset$ do
16:         $\mathcal{T}^m = \mathcal{T}^m \setminus \{i^m\}$
17:         $p^m = p^m + p_{i^m j^m}$
18:         $X_{i^m j^m} = 1$
19:         $i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
20:       Balance the assignment of reduce tasks (repeat lines
          15-19 for reduce tasks).
21:     else
22:       The code for $f < 1$ is similar to lines 15-20 and is not
          presented here.
23:   while $p^m + p^r + p_{i^m j^m} + p_{i^r j^r} \le D$ and $p^m + p_{i^m j^m} \le D^m$
      and $p^r + p_{i^r j^r} \le D^r$ and ($\mathcal{T}^m \neq \emptyset$ or $\mathcal{T}^r \neq \emptyset$)
```

After allocating the map and reduce tasks with the largest processing time, EMRSA-X assigns small map and reduce tasks while satisfying the deadline by calling ASSIGN-SMALL() (given in Algorithm 3). ASSIGN-SMALL() selects the smallest map task i, and based on the already assigned tasks and the remaining processing time of the slot, it decides if allocating task i is feasible or not (line 3). Then, it selects the smallest reduce task i, and checks the feasibility of its assignment (line 10).

Algorithm 3. ASSIGN-SMALL()

```
 1: {Assign small map tasks}
 2: $i = \text{argmin}_{t \in \mathcal{T}^m} p_{tj^m}$
 3: while $p^m + p^r + p_{ij^m} \le D$ and $p^m + p_{ij^m} \le D^m$
    and $\mathcal{T}^m \neq \emptyset$ do
 4:   $\mathcal{T}^m = \mathcal{T}^m \setminus \{i\}$
 5:   $p^m = p^m + p_{ij^m}$
 6:   $X_{ij^m} = 1$
 7:   $i = \text{argmin}_{t \in \mathcal{T}^m} p_{tj^m}$
 8: {Assign small reduce tasks}
 9: $i = \text{argmin}_{t \in \mathcal{T}^r} p_{tj^r}$
10: while $p^m + p^r + p_{ij^r} \le D$ and $p^m + p_{ij^m} \le D^r$ and $\mathcal{T}^r \neq \emptyset$ do
11:   $\mathcal{T}^r = \mathcal{T}^r \setminus \{i\}$
12:   $p^r = p^r + p_{ij^r}$
13:   $Y_{ij^r} = 1$
14:   $i = \text{argmin}_{t \in \mathcal{T}^r} p_{tj^r}$
```

The time complexity of EMRSA-X is $O(A(M + \log A) + B(R + \log B) + \min(A, B)(M \log M + R \log R))$, where A, B, M, and R are the number of map slots, the number of reduce slots, the number of map tasks, and the number of reduce tasks, respectively. The first two terms correspond to the running time of the two for loops in lines 4-5 and 6-8, while the third term corresponds to the running time of the while loop in lines 10-21.

3.1 Example

We now describe how EMRSA-II algorithm works by considering an example. We consider a job with two map tasks $\{t_1^m, t_2^m\}$ and two reduce tasks $\{t_1^r, t_2^r\}$ with a deadline of 12, and a data center with three map slots $\{a_1, a_2, a_3\}$ and two reduce slots $\{b_1, b_2\}$. The processing time and energy consumption of the map and reduce tasks are presented in Tables 1 and 2, respectively. For example, task t_1^m has 8 units of processing time and 8 units of energy consumption if it runs on map slot a_1 (i.e., $p_{11} = 8$ and $e_{11} = 8$). Then, we have $ecr^m = \{1, 2.5, 4.5\}$ and $ecr^r = \{3, 1.5\}$ for the map and reduce slots. EMRSA-II determines $\mathcal{Q}^m = \{a_1, a_2, a_3\}$ and $\mathcal{Q}^r = \{b_2, b_1\}$ (Algorithm 1, lines 1-8). Based on the priority queues, $\mathcal{Q}^m$ and $\mathcal{Q}^r$, the first map slot to take into account is a_1, and the first reduce slot is b_2. For these slots, the longest tasks are t_1^m and t_1^r, respectively (i.e., $X_{11} = 1$ and $Y_{12} = 1$). Based on the deadline, the algorithm cannot assign more tasks to these slots. Therefore, the deadlines for the map and reduce tasks are $D^m = 8$ and $D^r = 12 - 8 = 4$, respectively. That means, map tasks can be assigned to the other slots with the deadline of 8, and the reduce tasks can be assigned to the other slots from time 8 by the deadline of 12. The map tasks assignment is as follows. So far we have $X_{11} = 1$, the algorithm chooses the second map slot in $\mathcal{Q}^m$, and finds the longest task that has not been assigned to any slot yet. That means t_2^m is assigned to a_2 (i.e., $X_{22} = 1$). For the reduce tasks assignment, we already have $Y_{12} = 1$. The algorithm chooses the second reduce slot in $\mathcal{Q}^r$, and finds the longest task that has not been assigned to any slot yet. That means t_2^r is assigned to b_1 (i.e., $Y_{21} = 1$). This solution leads to a total energy consumption of 21 units, while satisfying the deadline constraint.

However, the solution that minimizes the makespan will select $X_{13} = 1$ and $X_{22} = 1$ to obtain a map makespan of 2 units, and will select $Y_{11} = 1$ and $Y_{22} = 1$ to obtain a reduce makespan of 2 units. This solution leads to a total makespan of 4 units with a total energy consumption of 26. Both approaches obtain schedules that meet the deadline. However, our proposed algorithm reduces the energy consumption by 19 percent. Note that the makespan for our approach is 11.

4 Experimental Results

We perform extensive experiments in order to investigate the properties of the proposed algorithms, EMRSA-I and EMRSA-II. We compare the performance of EMRSA-I and EMRSA-II with that of OPT, where OPT obtains the optimal solution minimizing the energy consumption. OPT is obtained by optimally solving the EMRS-IP problem (Equations (2) to (7)). Since OPT cannot find the optimal solutions in several cases due to the intractability of the problem, we present the results of the linear programming relaxation of EMRS-IP by changing the binary decision variables into continuous decision variables (Constraints (6) and (7)). The LP relaxation of EMRS-IP, called EMRS-LP, transforms an NP-hard optimization problem (EMRS-IP) into a related problem that is solvable in polynomial time. EMRS-LP gives a lower bound on the optimal solution of EMRS-IP by allowing partial assignments of each task to machines. Therefore, $OPT_{EMRS\text{-}LP} \leq OPT_{EMRS\text{-}IP}$, where $OPT_{EMRS\text{-}LP}$ is the optimal solution to EMRS-LP, and $OPT_{EMRS\text{-}IP}$ is the optimal solution to EMRS-IP.

However, such partial assignment of each task (obtained by solving the relaxation of EMRS-IP) is not a solution for the problem and cannot be used in practice. We only use the solution of the EMRS-LP (the relaxation of EMRS-IP) as a lower bound for EMRS-IP and compare it with the solutions obtained by the other algorithms. We denote by L-BOUND the algorithm that solves EMRS-LP and produces the lower bound on the solutions.

In addition, we present the results of minimizing the makespan, MSPAN, to show how far the current practice in MapReduce scheduling is from the optimal solutions that consider energy savings objectives. MSPAN is obtained by optimally solving the IP corresponding to the MapReduce makespan minimization problem (the same constraints as in ERMSA-IP, but the objective is makespan minimization). Since MSPAN cannot find the optimal solutions in several cases due to the intractability of the problem, we implemented a greedy algorithm for makespan minimization, called G-MSPAN. G-MSPAN schedules the tasks on the machines such that the processing time of all machines are balanced. It assigns longer tasks to faster machines to keep the balance.

TABLE 3
Selected HiBench Workloads

Category	Workload
Micro Benchmarks	TeraSort
Web Search	Page Rank
Machine Learning	K-means clustering

To analyze the performance of EMRSA-I and EMRSA-II, we present two classes of experiments, small-scale and large-scale. In the small-scale experiments, we compare the performance of EMRSA-I, EMRSA-II, OPT, and MSPAN for small MapReduce jobs. For large jobs, however, we cannot obtain the optimal results for OPT and MSPAN even after 24 hours, thus, we compare the performance of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN.

EMRSA-I, EMRSA-II, OPT, L-BOUND, MSPAN, and G-MSPAN algorithms are implemented in C++. OPT, MSPAN, and L-BOUND are implemented using APIs provided by IBM ILOG CPLEX Optimization Studio Multiplatform Multilingual eAssembly [44]. In this section, we describe the experimental setup and analyze the experimental results.

4.1 Experimental Setup

We performed extensive experiments on a Hadoop cluster of 64 processors and measured the energy and execution time for several MapReduce HiBench benchmark workloads [45]. HiBench is a comprehensive benchmark suite for Hadoop provided by Intel to characterize the performance of MapReduce based data analysis running in data centers. HiBench contains ten workloads, classified into four categories: Micro Benchmarks, Web Search, Machine Learning, and Analytical Query. We select three workloads, TeraSort, Page Rank, and K-means clustering, from different categories as shown in Table 3. The cluster is composed of four Intel nodes, with one node as a master. Two of the nodes have 24 GB memory, 16 2.4 GHz Intel processors, and a 1 TB Hard Drive. The other two nodes have 16 GB memory, 16 2.4 GHz Intel processors, and a 1 TB Hard Drive. The cluster has a total of 80 GB memory, 64 processors, 4 TB of storage, and network speed of 1Gbps. We set one map slot and one reduce slot per processor. Energy measurements were taken using Wattsup? PRO ES.Net Power meter. The input voltage is 100-250 Volts at 60 HZ and the max wattage is 1,800 Watts. The measurement accuracy is +/- 1.5 percent and the selected interval of time between records is one second.

We run and profiled several TeraSort, Page Rank, and K-means clustering workloads from the HiBench benchmark set. Each workload contains both map and reduce tasks. For each workload, we collect its start time, finish time, the consumed power and other performance metrics. We used 240 workloads for job profiling. We run only one job at a time, and collect the energy measurements and execution times. Since the reduce tasks execute only after the

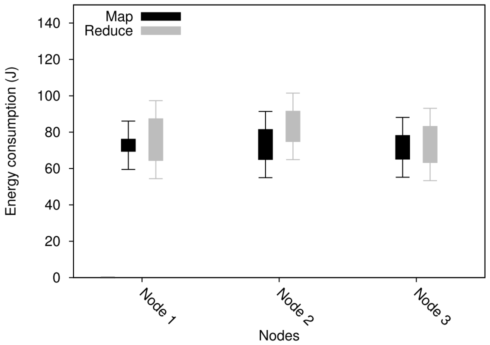


Fig. 1. Energy needs of tasks for the actual and simulated architectures.

execution of all map tasks is completed, we do not have overlaps between the map and reduce tasks. Based on the collected job profiles, we generated four small MapReduce jobs that we use in the small-scale experiments with the deadline of 250 seconds, and twenty four large MapReduce jobs, that we use in the large-scale experiments with the deadline of 1,500 seconds. Since for production jobs the choice of the deadline is at the latitude of the users, we select the deadlines specifically to obtain feasible schedules. The execution time and the energy consumption of the map and reduce tasks composing these jobs were generated from uniform distributions having as the averages the average energy consumption and the average execution time of the map and reduce tasks extracted from the jobs profiled in our experiments. Fig. 1 shows the energy needs of map and reduce tasks for the actual and simulated architecture. The energy consumption range of each node is shown as a filled box, where the bottom and the top of the box represent the minimum and the maximum energy consumption, respectively. For the simulated architecture, the energy consumption of each node is generated within a range whose boundaries are represented in the figure as horizontal lines. The simulation experiments are conducted on AMD 2.93 GHz hexa-core dual-processor systems with 90 GB of RAM which are part of the Wayne State Grid System.

4.2 Analysis of Results

4.2.1 Small-Scale Experiments

We analyze the performance of EMRSA-I, EMRSA-II, OPT, and MSPAN for four small MapReduce TeraSort jobs with 10,737,418 records, where the number of map tasks and reduce tasks are presented in Table 4. For example, the smallest job represented by $(48M, 48R)$ has 48 map tasks and 48 reduce tasks. Fig. 2a presents the energy consumption of the jobs scheduled by the four algorithms we consider. The results show that EMRSA-I and EMRSA-II obtain the assignments of map and reduce tasks with energy consumption close to the optimal solution, obtained by OPT. OPT, EMRSA-I, and EMRSA-II are able to schedule the tasks with an average of 41.0, 38.9, and 39.2 percent less energy consumption than that of MSPAN, respectively. For example, the total energy consumptions for workload $(48M, 48R)$ obtained by EMRSA-I, EMRSA-II, OPT, and MSPAN are 5,356, 5,396, 5,233, and 8,687 J, respectively. While it is desirable to use OPT as a scheduler to reduce cost, the slow execution of OPT makes it prohibitive to use in practice. In addition, it is practically impossible to use OPT when it comes to scheduling big data jobs due to its prohibitive runtime. EMRSA-I and EMRSA-II are very fast and practical alternatives for scheduling big data jobs, leading to 39 percent reduction in energy consumption. However, the energy consumption obtained by MSPAN is far from the optimal solution, making it not suitable for scheduling MapReduce jobs with the goal of minimizing the energy consumption.

TABLE 4
Terasort Workloads for the Small Scale Experiments

Workload	Map tasks	Reduce tasks
(48M, 48R)	48	48
(48M, 64R)	48	64
(64M, 48R)	64	48
(64M, 64R)	64	64

Fig. 2b presents the execution time of the algorithms. The results show that EMRSA-I and EMRSA-II find the assignments in significantly less amount of time than OPT and MSPAN. As shown in this figure, EMRSA-I and EMRSA-II obtain the solution in a time that is six orders of magnitude less than that of OPT. For example, the execution times of EMRSA-I, EMRSA-II, OPT, and MSPAN for the workload $(48M, 48R)$ are 0.001, 0.001, 673.7, and 839.3 seconds, respectively.

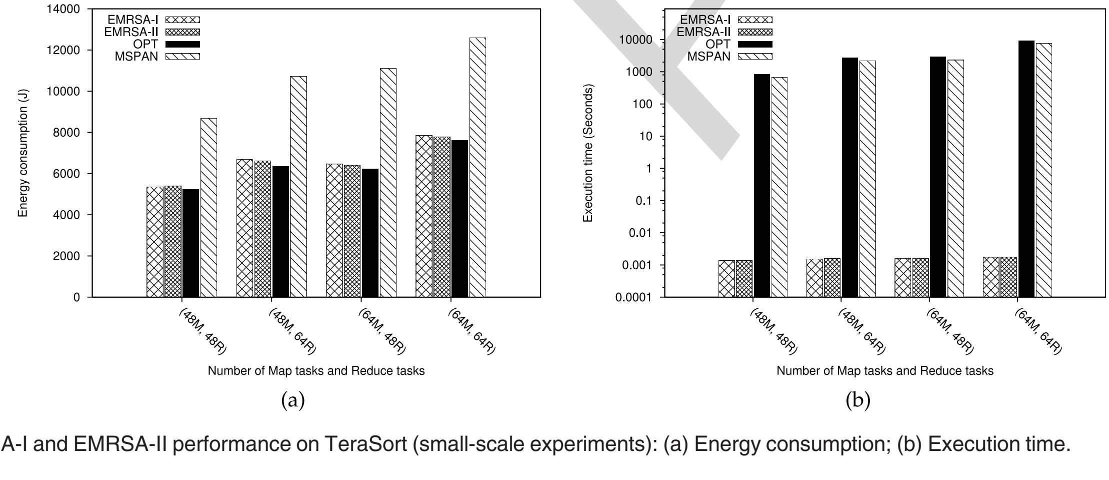
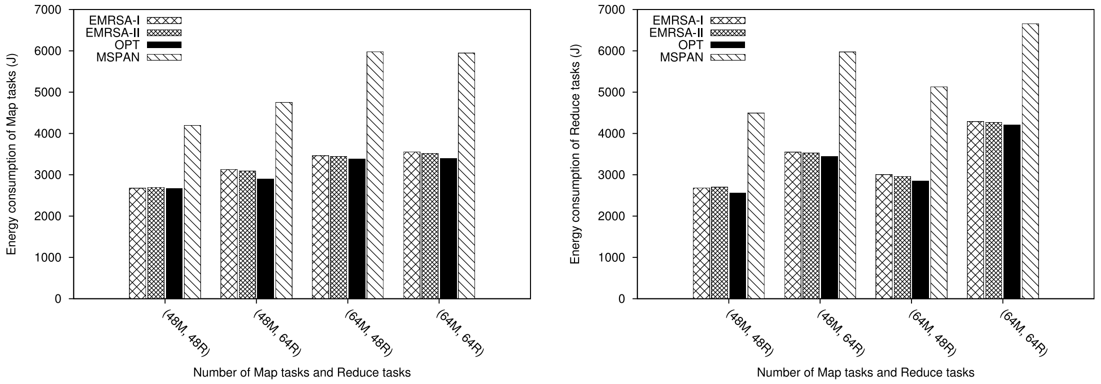


Fig. 2. EMRSA-I and EMRSA-II performance on TeraSort (small-scale experiments): (a) Energy consumption; (b) Execution time.



(a) (b)

Fig. 3. TeraSort energy consumption (small-scale experiments): (a) Map tasks; (b) Reduce tasks.

In Fig. 3, we present the energy consumption of map and reduce tasks in more details. When the number of reduce tasks is greater than the number of map tasks (e.g., workload $(48M, 64R)$), EMRSA-I and EMRSA-II capture more optimization opportunities for energy saving available for reduce tasks. In more detail, the energy consumptions of map tasks for workload $(48M, 64R)$ obtained by EMRSA-I, EMRSA-II, OPT, and MSPAN are 3,130, 3,090, 2,897, and 4,751 J, respectively, while the energy consumptions of reduce tasks for workload $(48M, 64R)$ are 3,547, 3,527, 3,448, and 5,972 J, respectively. However, when the workload has more map tasks than reduce tasks (e.g., workload $(64M, 48R)$), EMRSA-I and EMRSA-II save more energy for map tasks. The energy consumption for the map tasks of workload $(64M, 48R)$ obtained by employing EMRSA-X (shown in Fig. 3a) is closer to the optimal than the energy consumption for the reduce tasks (shown in Fig. 3b) for the same workload. This is due to the fact that for this workload, the load of the map tasks is greater than that of the reduce tasks, that is $f > 1$. For workload $(48M, 64R)$, where $f < 1$, EMRSA-X leads to an energy consumption closer to the optimal for the reduce tasks. This shows the effect of ratio f on the energy consumption.

4.2.2 Large-Scale Experiments

We analyze the performance of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN, for three types of benchmarks (TeraSort, Page Rank, and K-means clustering) considering eight large MapReduce jobs for each, where the number of map tasks and reduce tasks are given in Table 5.

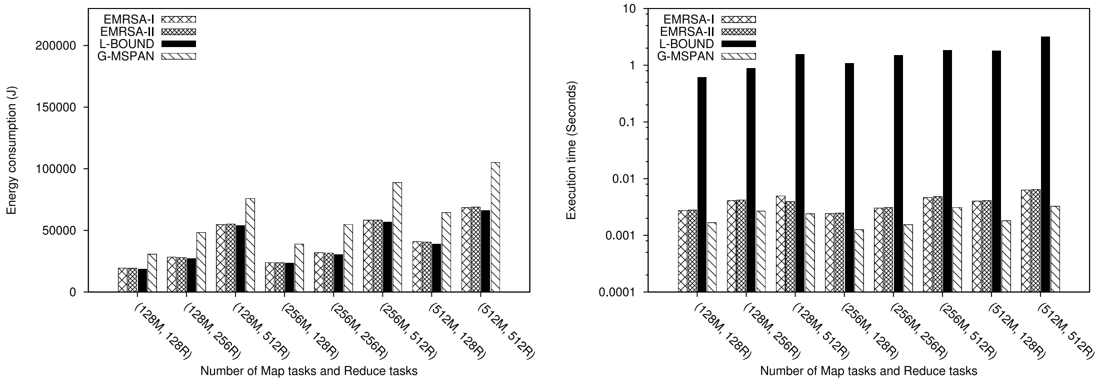
(i) Terasort. Fig. 4a shows the energy consumption of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN. This figure shows that the energy consumption obtained by EMRSA-I and EMRSA-II is very close to the lower bound for all cases, which in turn implies that EMRSA-I and EMRSA-II solutions are even closer to the optimal solutions. This shows the near-optimality of solutions obtained by EMRSA-I and EMRSA-II. In some cases EMRSA-I obtains better results. However, the results show that EMRSA-I and EMRSA-II are able to find schedules requiring an average of 35.6 and 35.8 percent less energy than that of those obtained by G-MSPAN, respectively. Such reduction in energy consumption can be a great incentive for data centers to incorporate EMRSA-I and EMRSA-II for scheduling MapReduce jobs to reduce their costs. Note that the amount of energy savings obtained by EMRSA-I and EMRSA-II in the large-scale experiments is compared with that obtained by the G-MSPAN. However, in the small-scale experiments, we presented the amount of energy savings of EMRSA-I and EMRSA-II compared to the optimal makespan minimization algorithm. As the total number of map and reduce tasks increases (from 256 to 1,024), the total amount of energy consumption of the workloads increases. In addition, this figure shows the sensitivity analysis on the number of tasks. By fixing the number of map tasks while increasing the number of reduce tasks, we observe an increase in the total energy consumption. For example, this behavior is shown for the first three workloads, where the number of map tasks is 128, and the number of reduce tasks is from 128 to 512.

TABLE 5
Workloads for the Large Scale Experiments

Workload	Map tasks	Reduce tasks
(128M, 128R)	128	128
(128M, 256R)	128	256
(128M, 512R)	128	512
(256M, 128R)	256	128
(256M, 256R)	256	256
(256M, 512R)	256	512
(512M, 128R)	512	128
(512M, 512R)	512	512

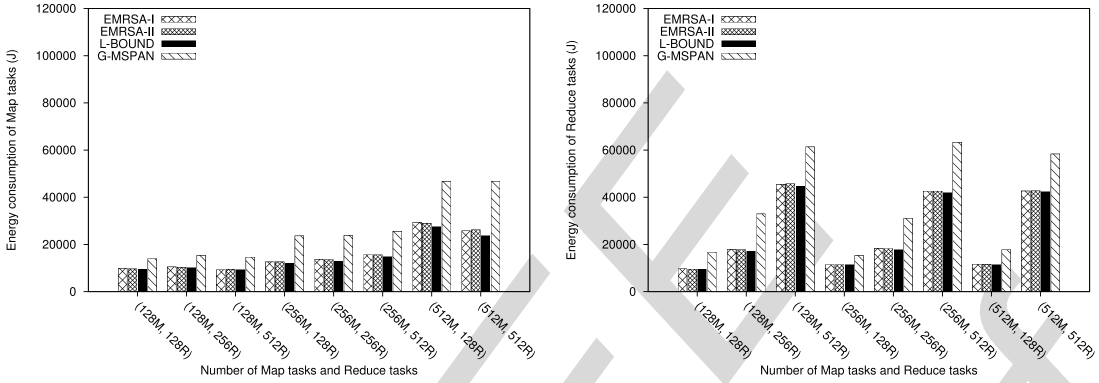
Fig. 4b shows the execution time of the algorithms. EMRSA-I, EMRSA-II, and G-MSPAN find the results in less than a second for all selected MapReduce jobs. Note that the lower bound results are obtained by solving the LP relaxation, not the EMRSA-IP. The execution time of L-BOUND presenting the LP relaxation results is polynomial. In addition, with the increase in the total number of map and reduce tasks, the execution time of L-BOUND increases. For example, the execution time of L-BOUND increases from workload (128 M, 128 R) to (128 M, 512 R). However, it decreases from workload (128 M, 512 R) to (256 M,128 R). The execution time of EMRSA-I and EMRSA-II follows the same behavior.

In Fig. 5, we present the energy consumption of map and reduce tasks separately in more detail. The results show that for both map and reduce tasks, the solutions obtained



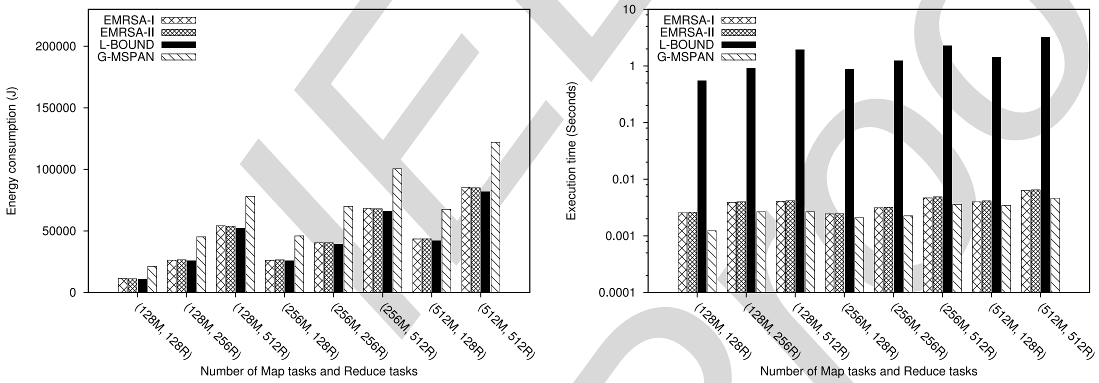
(a) (b)

Fig. 4. EMRSA-I and EMRSA-II performance on TeraSort (large-scale experiments): (a) Energy consumption; (b) Execution time.



(a) (b)

Fig. 5. TeraSort energy consumption (large-scale experiments): (a) Map tasks; (b) Reduce tasks.

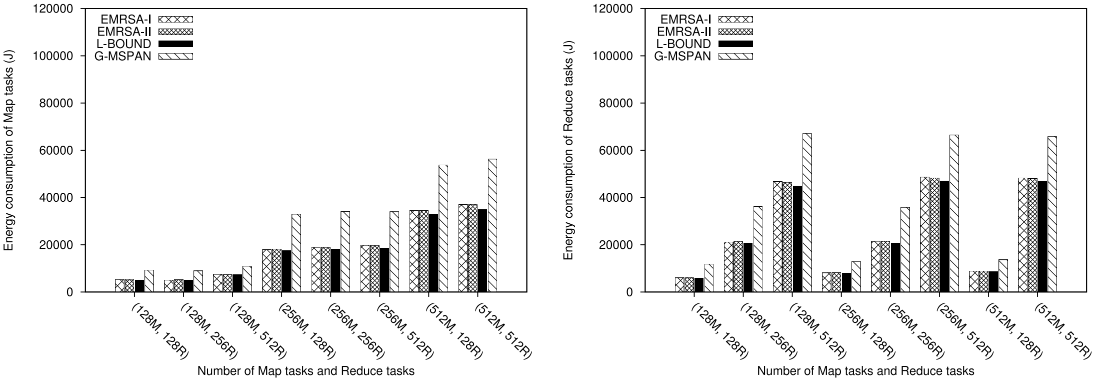


(a) (b)

Fig. 6. EMRSA-I and EMRSA-II performance on Page Rank (large-scale experiments): (a) Energy consumption; (b) Execution time.

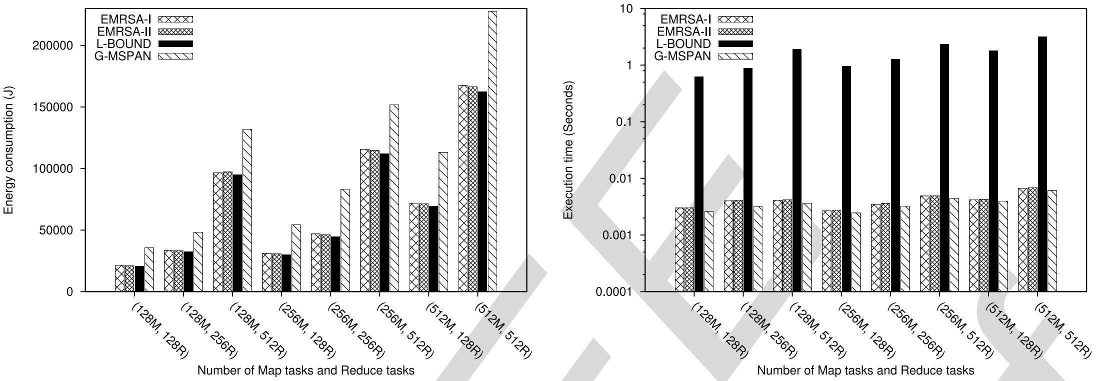
by EMRSA-I and EMRSA-II are very close to the lower bounds. In addition, we perform sensitivity analysis with respect to the number of map and reduce tasks. Fig. 5 shows how the energy consumption of map and reduce tasks changes by fixing the number of map tasks (e.g., to 128), and changing the number of reduce tasks (e.g., from 128 to 512). For example, for workloads (128 M, 128 R), (128 M, 256 R), and (128 M, 512 R), Fig. 5a shows that these workloads have almost the same energy consumption for their map tasks, which is expected as the number of map tasks are the same. However, Fig. 5b shows an exponential increase in the energy consumption of the reduce tasks for these workloads which is expected as the number of reduce tasks increases exponentially. For the sensitivity analysis of energy consumption with respect to number of reduce tasks, we analyze workloads (128 M, 128 R), (256 M, 128 R), and (512 M, 128 R). Fig. 5b shows that these workloads have almost the same energy consumption for their reduce tasks since the number of reduce tasks are the same. However, Fig. 5a shows that by increasing the number of map tasks, the energy consumption of the map tasks for these workload increases.

(ii) Page rank. Fig. 6a shows the energy consumption of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN. This figure shows that the obtained results by EMRSA-I and EMRSA-II are very close to the lower bounds obtained by



(a) (b)

Fig. 7. Page Rank energy consumption (large-scale experiments): (a) Map tasks; (b) Reduce tasks.



(a) (b)

Fig. 8. EMRSA-I and EMRSA-II performance on K-means clustering (large-scale experiments): (a) Energy consumption; (b) Execution time.

L-BOUND. The results show that EMRSA-I and EMRSA-II are able to find schedules requiring an average of 35.3 and 35.5 percent less energy than that of those obtained by G-MSPAN, respectively. The sensitivity analysis with respect to the number of tasks shows that by increasing the total number of map and reduce tasks, the energy consumption increases. For example, the total energy consumptions of $(256M, 128R)$ obtained by EMRSA-I, EMRSA-II, the L-BOUND, and G-MSPAN are 26,064, 26,364, 25,642.7, and 45,845 J, respectively, while the total energy consumptions of $(256M, 512R)$ are 68,388, 67,888, 65,832.3, and 100,426 J, respectively. For jobs with the same number of map tasks and the same number of reduce tasks, the energy consumption of Page Rank is similar to TeraSort energy consumption as shown in Fig. 4a.

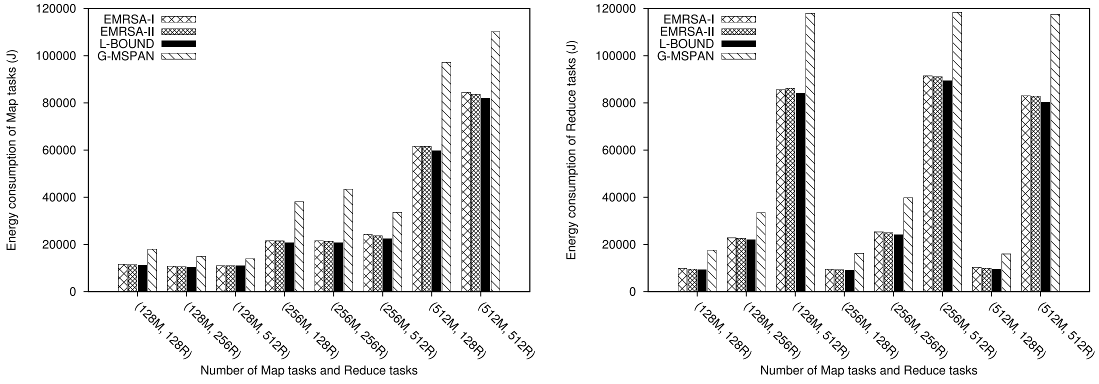
Fig. 6b shows the execution time of the algorithms. EMRSA-I, EMRSA-II, and G-MSPAN find the solutions very fast. With the increase in the total number of map and reduce tasks, the execution time of all algorithms increases. For example, the execution time of L-BOUND for workload $(512M, 128R)$ increases from 1.43 to 3.22 seconds for workload $(512M, 512R)$. In addition, the execution time of EMRSA-I and EMRSA-II for workload $(512M, 128R)$ increases from 0.004 to 0.006 seconds for workload $(512M, 512R)$.

Fig. 7 shows the energy consumption of map and reduce tasks separately in more detail, and it is similar to the results for TeraSort workloads. This figure shows the sensitivity analysis with respect to number of map and reduce tasks.

(iii) K-means clustering. Fig. 8a shows the energy consumption of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN. The results show that the obtained solutions by EMRSA-I and EMRSA-II are very close to the lower bounds obtained by L-BOUND. This figure shows that EMRSA-I and EMRSA-II are able to save an average of 30.9 and 31.4 percent energy compared to G-MSPAN, respectively. This figure also shows the sensitivity analysis on the total number of tasks, along with the detailed sensitivity analysis on the number of map and reduce separately. It confirms the above mentioned sensitivity analysis results for TeraSort and Page Rank. However, for similar workloads (i.e., having the same number of map tasks and the same number of reduce tasks) the energy consumption of K-means clustering is almost twice the energy consumption of Page Rank and TeraSort. This shows that K-means clustering tasks are more computationally complex than TeraSort and Page Rank, leading to consuming more energy.

In Fig. 8b, we present the execution time of the algorithms. The results show that EMRSA-I, EMRSA-II, and G-MSPAN find the solutions very fast. In Fig. 9, we present the energy consumption of map and reduce tasks separately in more details. The results show that for both map and reduce tasks, the obtained results by EMRSA-I and EMRSA-II are very close to the lower bounds. In addition, this figure shows the sensitivity analysis with respect to the number of map and reduce tasks.

From all the above results, we conclude that EMRSA-I and EMRSA-II obtain MapReduce job schedules with



(a) (b)

Fig. 9. K-means clustering energy consumption (large-scale experiments): (a) Map tasks; (b) Reduce tasks.

significantly lower energy consumption, and require small execution times, making them suitable candidates for scheduling big data applications in data centers. In addition, the schedules obtained by EMRSA-I and EMRSA-II provide energy savings close to the optimal. The results show that makespan minimization is not necessarily the best strategy to consider when scheduling MapReduce jobs for energy efficiency in data centers. This is due to the fact that data centers are obligated to deliver the requested services according to the SLA, where such agreement may provide significant optimization opportunities to reduce energy costs. Such reduction in energy costs is a great incentive for data centers to adopt our proposed scheduling algorithms.

5 Conclusion

Due to the increasing need for big data processing and the widespread adoption of MapReduce and its open source implementation Hadoop for such processing, improving MapReduce performance with energy saving objectives can have a significant impact in reducing energy consumption in data centers. In this paper, we show that there are significant optimization opportunities within the MapReduce framework in terms of reducing energy consumption. We proposed two energy-aware MapReduce scheduling algorithms, EMRSA-I and EMRSA-II, that schedule the individual tasks of a MapReduce job for energy efficiency while meeting the application deadline. Both proposed algorithms provide very fast solutions making them suitable for execution in real-time settings. We performed experiments on a Hadoop cluster to determine the energy consumption of several MapReduce benchmark applications such as TeraSort, Page Rank, and K-means clustering. We then used this data in an extensive simulation study to analyze the performance of EMRSA-I and EMRSA-II. The results showed that the proposed algorithms are capable of obtaining near optimal solutions leading to significant energy savings. In the future, we plan to design and implement a distributed scheduler for multiple MapReduce jobs with the primary focus on energy consumption.

Acknowledgments

This paper is a revised and extended version of [46] presented at the the 3rd IEEE International Congress on Big Data (BigData 2014). This research was supported, in part, by NSF grants DGE-0654014 and CNS-1116787.

References

[1] J. Koomey, "Growth in data center electricity use 2005 to 2010," vol. 1. Oakland, CA, USA: Analytics Press, Aug. 2011.

[2] J. Hamilton, "Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for internet-scale services," in *Proc. Conf. Innovative Data Syst. Res.*, 2009.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th USENIX Symp. Oper. Syst. Des. Implementation*, 2004, pp. 137–150.

[4] Hadoop. (2014) [Online]. Available: http://hadoop.apache.org/

[5] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user MapReduce clusters," Electrical Eng. Comput. Sci. Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-55, Apr. 2009.

[6] Apc. (2014) [Online]. Available: http://www.apc.com/

[7] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, "Resource-aware adaptive scheduling for mapreduce clusters," in *Proc. 12th ACM/IFIP/USENIX Int. Middleware Conf.*, 2011, pp. 187–207.

[8] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic resource inference and allocation for MapReduce environments," in *Proc. 8th ACM Int. Conf. Autonomic Comput.*, 2011, pp. 235–244.

[9] A. Verma, L. Cherkasova, and R. H. Campbell, "Two sides of a coin: Optimizing the schedule of MapReduce jobs to minimize their makespan and improve cluster performance," in *Proc. IEEE 20th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, 2012, pp. 11–18.

[10] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality-aware resource allocation for MapReduce in a cloud," in *Proc. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011.

[11] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós, "On scheduling in map-reduce and flow-shops," in *Proc. 23rd Annu. ACM Symp. Parallelism Algorithms Archit.*, 2011, pp. 289–298.

[12] H. Chang, M. S. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, "Scheduling in MapReduce-like systems for fast completion time," in *Proc. IEEE 30th Int. Conf. Comput. Commun.*, 2011, pp. 3074–3082.

[13] F. Chen, M. S. Kodialam, and T. V. Lakshman, "Joint scheduling of processing and shuffle phases in MapReduce systems," in *Proc. IEEE Conf. Comput. Commun.*, 2012, pp. 1143–1151.

[14] Y. Zheng, N. B. Shroff, and P. Sinha, "A new analytical technique for designing provably efficient MapReduce schedulers," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 1600–1608.

[15] T. J. Hacker and K. Mahadik, "Flexible resource allocation for reliable virtual cluster computing systems," in *Proc. Int. Conf. High Perform. Comp., Netw., Storage Anal.*, 2011, pp. 1–12.

[16] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for MapReduce in a cloud," *IEEE Trans. Parallel Distrib. Syst.*, no. 1, pp. 1, PrePrints, 2014.

[17] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative MapReduce," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.*, 2010, pp. 810–818.

[18] F. Tian and K. Chen, “Towards optimal resource provisioning for running MapReduce programs in public clouds,” in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 155–162.

[19] J. Zhan, L. Wang, X. Li, W. Shi, C. Weng, W. Zhang, and X. Zang, “Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers,” *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2155–2168, Nov. 2013.

[20] Y. Song, Y. Sun, and W. Shi, “A two-tiered on-demand resource allocation mechanism for VM-based data centers,” *IEEE Trans. Services Comput.*, vol. 6, no. 1, pp. 116–129, Jan. 2013.

[21] M. Nejad, L. Mashayekhy, and D. Grosu, “Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds,” *IEEE Trans. Parallel Distrib. Syst.*, vol. PP, no. 1, pp. 1, 2014.

[22] L. Mashayekhy, M. Nejad, and D. Grosu, “Cloud federations in the sky: Formation game and mechanism,” *IEEE Trans. Cloud Comput.*, vol. PP, no. 1, pp. 1, 2014.

[23] L. Mashayekhy, M. Nejad, D. Grosu, and A. V. Vasilakos, “Incentive-compatible online mechanisms for resource provisioning and allocation in clouds,” in *Proc. 7th IEEE Int. Conf. Cloud Comput.*, 2014.

[24] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, “Improving MapReduce performance in heterogeneous environments,” in *Proc. 8th USENIX Conf. Oper. Syst. Des. Implementation*, 2008, pp. 29–42.

[25] T. Sandholm and K. Lai, “MapReduce optimization using regulated dynamic prioritization,” in *Proc. 11th ACM Int. Conf. Meas. Model. Comput. Syst.*, 2009, pp. 299–310.

[26] X. Wang, D. Shen, G. Yu, T. Nie, and Y. Kou, “A throughput driven task scheduler for improving MapReduce performance in job-intensive environments,” in *Proc. IEEE 2nd Int. Congr. Big Data*, 2013, pp. 211–218.

[27] Z. Ren, X. Xu, M. Zhou, J. Wan, and W. Shi, “Workload analysis, implications, and optimization on a production Hadoop cluster: A case study on taobao,” *IEEE Trans. Services Comput.*, vol. 7, no. 2, pp. 307–321, Apr.–Jun. 2014.

[28] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell’Amico, and P. Michiardi, “HFSP: Size-based scheduling for Hadoop,” in *Proc. IEEE Int. Conf. Big Data*, 2013, pp. 51–59.

[29] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin, “Flex: A slot allocation scheduling optimizer for MapReduce workloads,” in *Proc. ACM/IFIP/USENIX 11th Int. Conf. Middleware*, 2010, pp. 1–20.

[30] T. Sandholm and K. Lai, “Dynamic proportional share scheduling in Hadoop,” in *Job Scheduling Strategies for Parallel Processing*. New York, NY, USA: Springer, 2010, pp. 110–131.

[31] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, “Job aware scheduling algorithm for MapReduce framework,” in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, 2011, pp. 724–729.

[32] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, “Maestro: Replica-aware map scheduling for MapReduce,” in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2012, pp. 435–442.

[33] S. Kurazumi, T. Tsumura, S. Saito, and H. Matsuo, “Dynamic processing slots scheduling for i/o intensive jobs of hadoop MapReduce,” in *Proc. IEEE 3rd Int. Conf. Netw. Comput.*, 2012, pp. 288–292.

[34] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt, “Evaluation and analysis of greenHDFS: A self-adaptive, energy-conserving variant of the Hadoop distributed file system,” in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, 2010, pp. 274–287.

[35] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, “Exploiting spatio-temporal tradeoffs for energy-aware MapReduce in the cloud,” *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1737–1751, Dec. 2012.

[36] J. Leverich and C. Kozyrakis, “On the energy (in) efficiency of Hadoop clusters,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 61–65, 2010.

[37] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, “Energy efficiency for large-scale MapReduce workloads with significant interactive analysis,” in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 43–56.

[38] N. Maheshwari, R. Nanduri, and V. Varma, “Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework,” *Future Generation Comput. Syst.*, vol. 28, no. 1, pp. 119–127, 2012.

[39] W. Lang and J. M. Patel, “Energy management for MapReduce clusters,” *Proc. VLDB Endowment*, vol. 3, nos. 1/2, pp. 129–139, 2010.

[40] T. Wirtz and R. Ge, “Improving MapReduce energy efficiency for computation intensive workloads,” in *Proc. IEEE Int. Green Comput. Conf. Workshops*, 2011, pp. 1–8.

[41] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, “GreenHadoop: Leveraging green energy in data-processing frameworks,” in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 57–70.

[42] M. A. Salehi, P. Radha Krishna, K. S. Deepak, and R. Buyya, “Preemption-aware energy management in virtualized data centers,” in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 844–851.

[43] S. Shen and J. Wang, “Stochastic modeling and approaches for managing energy footprints in cloud computing service,” *Service Sci.*, vol. 6, no. 1, pp. 15–33, 2014.

[44] IBM ILOG CPLEX V12.1 user’s manual. (2014). [Online]. Available: ftp://public.dhe.ibm.com/software/websphere/ilog/docs/op-timization/cplex/ps_usrmancplex.pdf

[45] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, “The HiBench benchmark suite: Characterization of the MapReduce-based data analysis,” in *Proc. IEEE 26th Conf. Data Eng. Workshops*, 2010, pp. 41–51.

[46] L. Mashayekhy, M. Nejad, D. Grosu, D. Lu, and W. Shi, “Energy-aware scheduling of MapReduce jobs,” in *Proc. IEEE 3rd Int. Congr. Big Data*, 2014.

Lena Mashayekhy received the BSc degree in computer engineering-software from the Iran University of Science and Technology, and the MSc degree from the University of Isfahan. She is currently working toward the PhD degree in computer science at Wayne State University, Detroit, Michigan. She has published more than 20 peer-reviewed papers in venues such as *IEEE Transactions on Parallel and Distributed Systems*, *IEEE BigData*, *IEEE CLOUD*, and ICPP. Her research interests include distributed systems, cloud computing, big data analytics, game theory, and optimization. She is a student member of the IEEE, ACM, and the IEEE Computer Society.

Mahyar Movahed Nejad received the BSc degree in mathematics from the Iran University of Science and Technology and the MSc degree in socio-economic systems engineering from the Mazandaran University of Science and Technology. He is currently working toward the MSc degree in computer science, and the PhD degree in industrial and systems engineering at Wayne State University, Detroit. His research interests include cloud computing, big data analytics, game theory, network optimization, and integer programming. His papers appeared in journals such as *IEEE Transactions on Parallel and Distributed Systems*. He is a student member of the IEEE and the INFORMS.

Daniel Grosu received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iaşi, Romania, in 1994, and the MSc and PhD degrees in computer science from the University of Texas at San Antonio, in 2002 and 2003, respectively. He is currently an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed systems, cloud computing, parallel algorithms, resource allocation, computer security, and topics at the border of computer science, game theory and economics. He has published more than ninety peer-reviewed papers in the above areas. He has served on the program and steering committees of several international meetings in parallel and distributed computing. He is a senior member of the IEEE, ACM, and the IEEE Computer Society.

Quan Zhang received the BSc degree in information security from Tongji University, Shanghai, China. He is currently working toward the PhD degree in computer science at Wayne State University, Detroit, Michigan. His research interests include distributed systems, cloud computing, and energy efficient computing system. He is a student member of the IEEE.

Weisong Shi received the BE degree from Xidian University in 1995, and the PhD degree from the Chinese Academy of Sciences in 2000, both in computer engineering. He is a professor of computer science at Wayne State University, where he leads the Mobile and Internet Systems Laboratory. His research interests include computer systems, sustainable computing, mobile computing, and smart health. He has published more than 140 peer-reviewed journal and conference papers and has an H-index of 30. He is the chair of the IEEE CS Technical Committee on the Internet, and serves on the editorial board of *IEEE Internet Computing*, Elsevier Sustainable Computing, *Journal of Computer Science and Technology (JCST)*, and *International Journal of Sensor Networks*. He received the National Outstanding PhD Dissertation Award of China (2002) and the National Science Foundation (NSF) CAREER Award (2007), Wayne State University Career Development Chair Award (2009), and the Best Paper Award of ICWE04, IEEE IPDPS05, HPCChina'12, and IEEE IISWC'12. He is a senior member of the IEEE, ACM and a member of the USENIX.