Energy-aware Scheduling of MapReduce Jobs

Lena Mashayekhy, Mahyar Movahed Nejad, Daniel Grosu, Dajun Lu, Weisong Shi
Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
{mlena, mahyar, dgrosu, ei8449, weisong}@wayne.edu

***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 large fraction of the data center's overall costs. Therefore minimizing the energy consumption when executing MapReduce jobs 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 MapReduce jobs as an Integer Program. We then propose a greedy algorithm, called Energy-aware MapReduce Scheduling Algorithm (EMRSA), that finds the assignments of map and reduce tasks to the machine slots in order to minimize the energy consumed when executing the application. We perform experiments on a large Hadoop cluster to determine the energy consumption of several MapReduce benchmark applications, and then use this data in an extensive simulation study to characterize the performance of the proposed algorithm. The results show that EMRSA is able to find job schedules consuming 40% less energy on average than the schedules obtained by a common practice scheduler that minimizes the makespan.**

***Keywords*-MapReduce; big data; minimizing energy consumption; scheduling.**

I. Introduction

Electricity used in US data centers in 2010 accounted for about 2% of total electricity used nationwide [1]. In addition, the energy consumed by the data centers is growing at over 15% annually, and the energy costs make up about 42% 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. On the other hand, 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 a MapReduce scheduling algorithm that improves the energy efficiency of running each individual application, while satisfying the service level agreement (SLA). Our proposed scheduling algorithm can be easily incorporated and deployed within the existing Hadoop systems.

In most of the cases, processing bigdata 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 power-meter 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 algorithm schedules 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 algorithm captures such opportunities and significantly reduces the MapReduce energy costs, while satisfying the SLA.

Our Contribution. To the best of our knowledge this is the first study that designs an algorithm for detailed task placement of a MapReduce job to machines with the primary focus on minimizing the energy consumption. Our proposed algorithm 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 a greedy algorithm, called Energy-aware MapReduce Scheduling Algorithm (EMRSA). EMRSA provides very fast solutions making it suitable for deployment in real production MapReduce clusters. The time complexity of EMRSA is polynomial in the number of map and reduce slots, the number of map tasks, and the number of reduce tasks, respectively. We perform experiments on a large Hadoop cluster to determine the energy consumption of several MapReduce benchmark applications, and then use this data in an extensive simulation study to characterize the performance of the proposed algorithm. The results show that EMRSA is capable of finding close to optimal solutions.

Related Work. There exists an extensive body of research on resource allocation and scheduling in data centers and clouds that does not consider the energy efficiency (e.g., [11], [15], [16]). However, in this literature review, we only discuss briefly the studies that are directly related to energy savings in data centers. Kaushik et al. [17] 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. [18] 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 [19] proposed a method for cluster energy management for 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. [20] 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. [21] proposed an algorithm that dynamically reconfigures clusters by scaling up and down the number of nodes based on the cluster utilization. Land and Patel [22] 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 [23] 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. [24] 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. Li et al. [25] studied the performance of three hypervisors by running several MapReduce benchmarks. Some of the considered performance criteria are completion time, CPU, and disk utilizations. However, they did not focus on energy and power consumption as the performance criteria. 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. However, 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 algorithm considers the significant energy consumption differences of different task placements on machines, and finds an energy efficient assignment of tasks to machines.

Organization. The rest of the paper is organized as follows. In Section II, we describe the problem of scheduling MapReduce jobs for energy efficiency. In Section III, we present our proposed algorithm. In Section IV, we evaluate the algorithm by extensive experiments. In Section V, we summarize our results and present possible directions for future research.

II. 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 map and 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 speed and the energy consumption of the machines may not be the same. We denote by e_{ij} the energy consumption of slot $j \in \{\mathcal{A}, \mathcal{B}\}$ when executing task $i \in \{\mathcal{M}, \mathcal{R}\}$ and by p_{ij} the processing time 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 $\delta_{ti}, \forall t, i \in \mathcal{M} \cup \mathcal{R}$, 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:

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

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

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

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

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

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 to $\max_{\forall j\in\mathcal{A}} \sum_{i\in\mathcal{M}} p_{ij}X_{ij} + \max_{\forall j'\in\mathcal{B}} \sum_{i\in\mathcal{R}} p_{ij'}Y_{ij'} \le D$. 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.

III. Energy-aware MapReduce Scheduling Algorithm

We design a greedy algorithm, called Energy-aware MapReduce Scheduling Algorithm (EMRSA), that finds energy-efficient schedules for MapReduce jobs. A key challenge when designing the algorithm is that the user only specifies the deadline for the job without specifying a deadline for the map phase. However, since the reduce tasks are dependent on the map tasks, the data center has to determine a reasonable deadline for the map tasks with respect to the availability of the map slots in the data center in order to utilize its resources efficiently. The proposed algorithm finds the assignments of map tasks to the map slots satisfying the determined map deadline. Finally, EMRSA finds the assignments of reduce tasks to the reduce slots satisfying the deadline, where all the reduce tasks start after the map deadline.

The design of the algorithm requires a metric that characterizes the energy consumption of each machine and induces a order relation among the machines. We define such metric, called *energy consumption rate* of a slot j, as follows:

$$ecr_j^m = \frac{\sum_{\forall i\in\mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}, \forall j \in \mathcal{A} \tag{10}$$

$$ecr_j^r = \frac{\sum_{\forall i\in\mathcal{R}} \frac{e_{ij}}{p_{ij}}}{R}, \forall j \in \mathcal{B} \tag{11}$$

where ecr_j^m and ecr_j^r represent the energy consumption rate of map slot j and reduce slot j, respectively. A lower ecr_j^m means a higher priority for the slot j to which a task is assigned.

The Energy-aware MapReduce Scheduling Algorithm (EMRSA) is given in Algorithm 1. In the first phase, EMRSA 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

Algorithm 1 EMRSA

```
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:      $ecr^m_j = \frac{\sum_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}$
5:      $\mathcal{Q}^m$.enqueue$(j, ecr^m_j)$
6:  for all $j \in \mathcal{B}$ do
7:      $ecr^r_j = \frac{\sum_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}}{R}$
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
17:         break
18:     $i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
19:     $i^r = \text{argmax}_{t \in \mathcal{T}^r} p_{tj^r}$
20:     $p^m = 0$; $p^r = 0$
21:     if $D^m = \infty$ and $p_{i^m j^m} + p_{i^r j^r} > D$ then
22:         No feasible schedule
23:         return
24:     while $p^m + p^r + p_{i^m j^m} + p_{i^r j^r} \leq D$ and $p^m + p_{i^m j^m} \leq D^m$
        and $p^r + p_{i^r j^r} \leq D^r$ and $(\mathcal{T}^m \neq \emptyset$ or $\mathcal{T}^r \neq \emptyset)$ do
25:         $\mathcal{T}^m = \mathcal{T}^m \setminus \{i^m\}$
26:         $\mathcal{T}^r = \mathcal{T}^r \setminus \{i^r\}$
27:         $p^m = p^m + p_{i^m j^m}$
28:         $p^r = p^r + p_{i^r j^r}$
29:         $X_{i^m j^m} = 1$
30:         $Y_{i^r j^r} = 1$
31:         if $f > 1$ then
32:             $i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
33:             while $\frac{p^m + p_{i^m j^m}}{p^r} < f$ and $p^m + p^r + p_{i^m j^m} \leq D$ and
                $p^m + p_{i^m j^m} \leq D^m$ and $\mathcal{T}^m \neq \emptyset$ do
34:                 $\mathcal{T}^m = \mathcal{T}^m \setminus \{i^m\}$
35:                 $p^m = p^m + p_{i^m j^m}$
36:                 $X_{i^m j^m} = 1$
37:                 $i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
38:             Balance the assignment of reduce tasks (repeat lines 32-37
                for reduce tasks).
39:         else
40:             The code for $f < 1$ is similar to lines 32-38 and is not
                presented here.
41:     $i = \text{argmin}_{t \in \mathcal{T}^m} p_{tj^m}$
42:     while $p^m + p^r + p_{ij^m} \leq D$ and $p^m + p_{ij^m} \leq D^m$ and $\mathcal{T}^m \neq \emptyset$
        do
43:         $\mathcal{T}^m = \mathcal{T}^m \setminus \{i\}$
44:         $p^m = p^m + p_{ij^m}$
45:         $X_{ij^m} = 1$
46:         $i = \text{argmin}_{t \in \mathcal{T}^m} p_{tj^m}$
47:     Assign small reduce tasks (repeat lines 41-46 for reduce tasks).
48:     if $D^m = \infty$ then
49:         $D^m = D - p^r$
50:         $D^r = D - D^m$
51: if $\mathcal{T}^m \neq \emptyset$ or $\mathcal{T}^r \neq \emptyset$ then
52:     No feasible schedule
53:     return
54: Output: $X, Y$
```

consumption rates (lines 1-8). EMRSA initializes the deadlines for map tasks, D^m, and reduce tasks, D^r, to infinity. In each iteration, the algorithm chooses the slots with the lowest energy consumption rates (i.e., j^m and j^r) from the priority queues. For these slots, the ratio of processing time of map tasks to that of the reduce tasks, denoted by f, is calculated (line 13). This ratio is used in the task assignment process in each iteration of the algorithm. Then, EMRSA sorts the unassigned map and reduce tasks based on their processing time on the selected slots (lines 14-15). It selects the longest map task i^m and reduce task i^r from the sorted sets $\mathcal{T}^m$ and $\mathcal{T}^r$, respectively (lines 18-19). 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 (lines 21-22). If the assignment of map task i^m and reduce task i^r is feasible (line 24), the algorithm continues to select tasks from $\mathcal{T}^m$ and $\mathcal{T}^r$, and updates the variables accordingly (lines 25-30). To keep the assignments of the tasks in alignment with the ratio of processing time f, the algorithm 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 32-37). If the ratio of the current assignment is greater than f, the algorithm assigns more reduce tasks to balance the allocated processing time (lines 38). After allocating the map and reduce tasks with the largest processing time, the algorithm assigns small map and reduce tasks while satisfying the deadline (lines 41-47). At the end of the first iteration, the algorithm sets the map and reduce deadlines based on the allocated tasks (lines 48-50). The time complexity of EMRSA is polynomial in the number of map slots, the number of reduce slots, the number of map tasks, and the number of reduce tasks, respectively.

A. Example

We now describe how the EMRSA algorithm works by considering an example. We consider a job with 2 map tasks $\{t^m_1, t^m_2\}$ and 2 reduce tasks $\{t^r_1, t^r_2\}$ with a deadline of 12, and a data center with 3 map slots $\{a_1, a_2, a_3\}$ and 2 reduce slots $\{b_1, b_2\}$. The processing time and energy consumption of the map and reduce tasks are presented in Table I and Table II, respectively. For example, task t^m_1 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. After the sorting step, $\mathcal{Q}^m = \{a_1, a_2, a_3\}$ and $\mathcal{Q}^r = \{b_2, b_1\}$. Based on the sorted sets of slots, $\mathcal{Q}^m$ 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^m_1 and t^r_1, 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

Table I: 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 II: 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

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%. Note that the makespan for our approach is 11.

IV. Experimental Results

We perform extensive experiments in order to investigate the properties of the proposed algorithm, EMRSA. We compare the performance of EMRSA with that of other two algorithms: (i) OPT, that obtains the optimal solution minimizing the energy consumption; and (ii) MSPAN, that obtains the optimal solution minimizing the makespan. OPT is obtained by optimally solving the EMRSA-IP problem (Equations (2) to (7)), while MSPAN is obtained by optimally solving the IP corresponding to the MapReduce makespan minimization problem (same constraints as in ERMSA-IP, but the objective is makespan minimization). Both OPT and MSPAN were implemented using IBM ILOG CPLEX Optimization Studio Multiplatform Multilingual eAssembly. For large jobs, however, CPLEX could not find the optimal results for OPT and MSPAN even after 3 hours. To analyze the results of EMRSA, we present two classes of experiments, small-scale and large-scale. In the small-scale experiments, we compare the results of OPT, MSPAN, and EMRSA for small MapReduce jobs. Since we cannot obtain the optimal results for large MapReduce jobs, 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. EMRSA, OPT, MSPAN, and G-MSPAN algorithms are implemented in C++ and the experiments are conducted on AMD 2.93GHz hexa-core dual-processor systems with 90GB of RAM which are part of the Wayne State Grid System. In this section, we describe the experimental setup and analyze the experimental results.

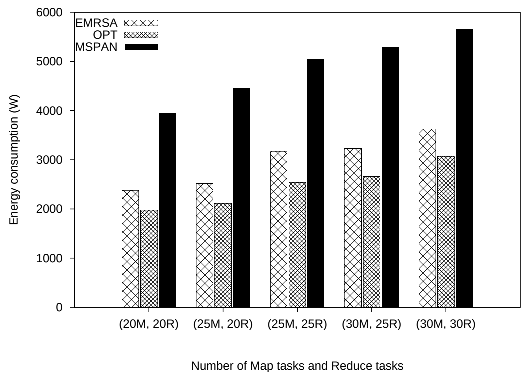
Table III: Terasort configurations for job profiling.

Workload	Records	Map tasks	Reduce tasks
I	64,424,509	16	28
II	257,698,037	64	112
III	515,396,075	128	224
IV	773,094,112	192	336

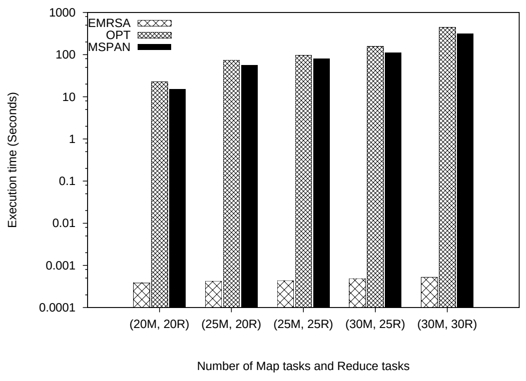
A. Experimental Setup

We performed extensive experiments on a large Hadoop cluster of 224 processors and measure the energy and execution time for several MapReduce benchmarks [27]. The cluster is composed of 10 nodes, two racks of four Intel nodes and one rack of two AMD nodes. The Intel nodes have 96GB memory, 12 2.96GHZ Intel processors, and a 600GB SAS 15K Hard Drive. The AMD nodes have 128GB memory, 64 2.6GHZ AMD processors, and a 600GB SAS 15K Hard Drive. The cluster has a total of 1TB memory, 224 processors, and 6TB of storage. Energy measurements were taken using APC AP8641 PDU Power Strip that provides real-time remote monitoring of each outlet.

We run and profiled several Terasort workloads from the HiBench benchmark set [28]. HiBench is a comprehensive benchmark provided by Intel to characterize the performance of MapReduce based data analysis running in data centers. The Terasort workload contains both map and reduce tasks. We used four Terasort workloads for job profiling, with the configurations presented in Table III. For each workload, we collect its start time, finish time, the consumed power and other performance metrics. For example for workload III, the map tasks have a total energy consumption of 3,350 W, while the reduce tasks have a total energy consumption of 2,800 W. Based on the collected job profiles, we generated five small MapReduce jobs that we use in the small-scale experiments, and four large MapReduce jobs, that we use in the large-scale experiments. The execution time and the

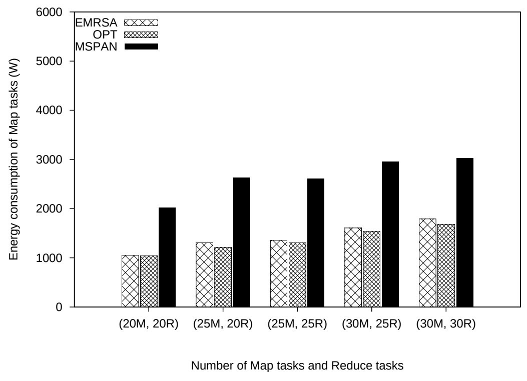


(a)

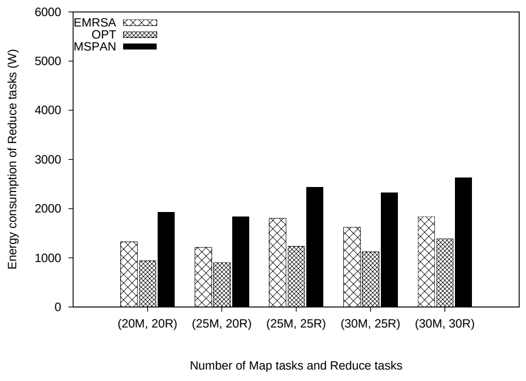


(b)

Figure 1: EMRSA performance (small-scale experiments): (a) Energy Consumption; (b) Execution time.



(a)



(b)

Figure 2: Energy Consumption (small-scale experiments): (a) Map tasks; (b) Reduce tasks.

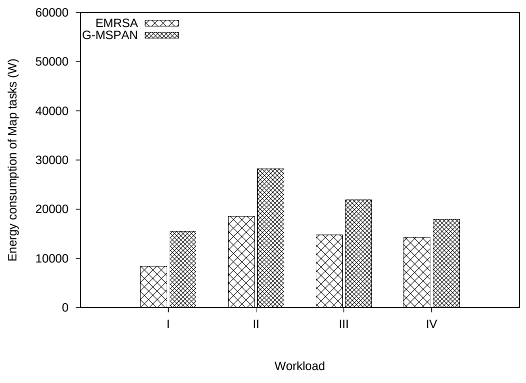
energy consumption of the map and reduce tasks composing these jobs were generated from uniform distributions having as the averages the average energy consumption and the average execution time of the map and reduce tasks extracted from the job profiled in our experiments.

B. Analysis of Results

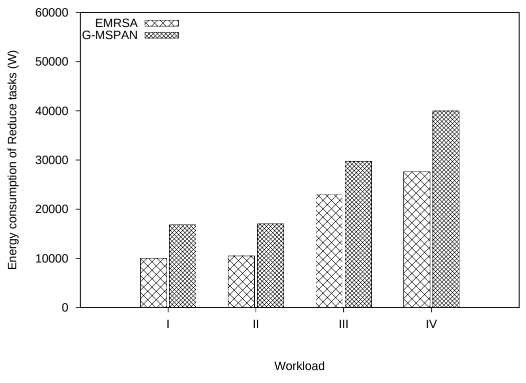
1) Small-scale experiments: We analyze the performance of EMRSA, OPT, and MSPAN for five small MapReduce jobs, where the number of map tasks and reduce tasks are within a range of 20 to 30 tasks. For example, the smallest job represented by $(20M, 20R)$ has 20 map tasks and 20 reduce tasks. We consider that the jobs are executed on a cluster configured with 20 map and 20 reduce slots. Fig. 1a represents the energy consumption of the jobs scheduled by the three algorithms we consider. The results show that EMRSA obtains the assignments of map and reduce tasks with energy consumption close to the optimal solution, obtained by OPT. OPT and EMRSA are able to schedule the tasks with an average 49% and 40% less energy consumption that that of MSPAN, 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 bigdata jobs due to its prohibitive runtime. EMRSA is very fast and a practical alternative for scheduling bigdata jobs, leading to 40% 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.

Fig. 1b represents the execution time of OPT and EMRSA. The results show that EMRSA finds the assignments in significantly less amount of time than OPT and MSPAN. As shown in this figure, EMRSA obtains the solution in a time that is five orders of magnitude less than that of OPT.

In Fig. 2, we present the energy consumption of map and reduce tasks in more details. The energy consumption for the map tasks scheduled by EMRSA is closer to the optimal than the energy consumption for the reduce tasks. This is due to the fact that for the selected small jobs, the load of the map tasks is greater or equal to that of the reduce tasks. In the next subsection, we show how EMRSA performs when the



(a)



(b)

Figure 3: Energy Consumption (large-scale experiments): (a) Map tasks; (b) Reduce tasks.

Table IV: Workloads for the large scale experiments.

Workload	Map tasks	Reduce tasks
I	160	200
II	300	200
III	250	400
IV	200	500

load of the reduce tasks is greater than that of the map tasks.

2) Large-scale experiments: We analyze the performance of EMRSA and the greedy Makespan, called G-MSPAN, for four large MapReduce jobs, where the number of map tasks and reduce tasks are presented in Table IV. We consider that the cluster is configured with 128 map and 128 reduce slots. Fig. 4 shows the energy consumption of EMRSA and G-MSPAN. The results show that EMRSA is able to find schedules requiring an average of 32% less energy than that of those obtained by G-MSPAN. Such reduction in energy consumption can be a great incentive for data centers to incorporate EMRSA for scheduling MapReduce jobs to reduce their costs. Note that the amount of energy savings obtained by EMRSA 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 compared to the optimal makespan minimization algorithm. Both EMRSA and G-MSPAN find the results in less than a second for all selected MapReduce jobs.

In Fig. 3, we present the energy consumption of map and reduce tasks separately in more detail. When the load of the reduce tasks is higher than that of the map tasks (e.g., workload IV), EMRSA captures more optimization opportunities available for reduce tasks for energy saving. When the load of the map tasks is higher than that of reduce tasks (e.g., workload I), EMRSA captures more optimization opportunities available for map tasks for energy saving.

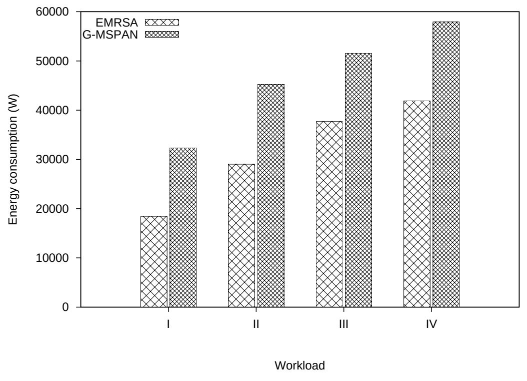


Figure 4: Energy Consumption (large-scale experiments).

From all the above results, we conclude that EMRSA obtains MapReduce job schedules with significantly less energy consumption, and requires small execution times, making it a suitable candidate for scheduling bigdata applications in data centers. In addition, the schedules obtained by EMRSA 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 algorithm.

V. Conclusion

We proposed an Energy-aware MapReduce Scheduling Algorithm, EMRSA, that schedules the individual tasks of a MapReduce application for energy efficiency while meeting the application deadline. EMRSA provides very fast solutions making it suitable for execution in real-time settings. We performed experiments on a large Hadoop cluster to determine the energy consumption of several MapReduce benchmark applications, and then used this data in an extensive simulation study to analyze the performance of EMRSA. The results showed that EMRSA is capable of obtaining significant energy savings compared to the standard makespan minimization algorithms. For the future, we plan to design and implement a scheduler for multiple MapReduce jobs with the primary focus of energy consumption.

Acknowledgment

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," *Oakland, CA: Analytics Press. August*, vol. 1, 2011.

[2] J. Hamilton, "Cooperative expendable micro-slice servers (cems): low cost, low power servers for internet-scale services," in *Proc. of the Conf. on Innovative Data Systems Research*, 2009.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. of the 6th USENIX Symposium on Operating System Design and Implementation*, 2004, pp. 137–150.

[4] Hadoop. [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," UC Berkeley, Tech. Rep. Technical Report UCB/EECS-2009-55, April 2009.

[6] Apc. [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. of the 12th ACM/IFIP/USENIX International Middleware Conference*, 2011, pp. 187–207.

[8] A. Verma, L. Cherkasova, and R. H. Campbell, "Aria: automatic resource inference and allocation for mapreduce environments," in *Proc. of the 8th International Conference on Autonomic Computing*, 2011, pp. 235–244.

[9] ——, "Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance," in *Proc. of the 20th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 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. of the Conference on High Performance Computing Networking, Storage and Analysis, SC 2011*, 2011.

[11] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós, "On scheduling in map-reduce and flow-shops," in *Proc. of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 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. of the 30th IEEE International Conference on Computer Communications*, 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. of the IEEE INFOCOM*, 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. of the IEEE INFOCOM*, 2013, pp. 1600–1608.

[15] M. M. Nejad, L. Mashayekhy, and D. Grosu, "A family of truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," in *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, 2013, pp. 188–195.

[16] L. Mashayekhy, M. M. Nejad, and D. Grosu, "A truthful approximation mechanism for autonomic virtual machine provisioning and allocation in clouds," in *Proc. of the ACM Cloud and Autonomic Computing Conf.*, 2013, pp. 1–10.

[17] 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. of the 2nd IEEE International Conf. on Cloud Computing Technology and Science*, 2010, pp. 274–287.

[18] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, "Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud," *IEEE Transactions on Computers*, pp. 1737–1751, 2012.

[19] J. Leverich and C. Kozyrakis, "On the energy (in) efficiency of hadoop clusters," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 61–65, 2010.

[20] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale mapreduce workloads with significant interactive analysis," in *Proc. of the 7th ACM European Conf. on Computer Systems*, 2012, pp. 43–56.

[21] N. Maheshwari, R. Nanduri, and V. Varma, "Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 119–127, 2012.

[22] W. Lang and J. M. Patel, "Energy management for mapreduce clusters," *Proc. of the VLDB Endowment*, vol. 3, no. 1-2, pp. 129–139, 2010.

[23] T. Wirtz and R. Ge, "Improving mapreduce energy efficiency for computation intensive workloads," in *Proc. of the IEEE International Green Computing Conference and Workshops*, 2011, pp. 1–8.

[24] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: leveraging green energy in data-processing frameworks," in *Proc. of the 7th ACM European Conf. on Computer Systems*, 2012, pp. 57–70.

[25] J. Li, Q. Wang, D. Jayasinghe, J. Park, T. Zhu, and C. Pu, "Performance overhead among three hypervisors: An experimental study using hadoop benchmarks," in *Proc. of the 2nd IEEE International Congress on Big Data*, 2013, pp. 9–16.

[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. of the 2nd IEEE International Congress on Big Data*, 2013, pp. 211–218.

[27] D. Lu, "Pulse: A framework for analyzing power related information in data centers," Master's thesis, Computer Science Department, Wayne State University, Detroit, 2014.

[28] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Proc. of the IEEE 26th Conf. on Data Engineering Workshops*, 2010, pp. 41–51.