International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 5 145 – 150

Proposed Energy Aware Scheduling Algorithm in Data Center by using MapReduce

Dr. M. Sughasiny
Assistant Professor, Department of Computer Science
Srimad Andavan Arts and Science College (Autonomous)
Trichy, Tamilnadu, India
sughasiny5.cs@gmail.com

G. Murali
Research Scholar, Department of Computer Science
Srimad Andavan Arts and Science College (Autonomous)
Trichy, Tamilnadu, India
gmuralimphil.cs@gmail.com

***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. 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% less energy on average than the schedules obtained by a common practice scheduler that minimizes the makespan.

Keywords*-Big Data, Hadoop, MapReduce, Energy Aware Scheduling Algorihtms, Terasort, Page Rank, Machine Learning, Web Search, Micro Benchmarks*

*****

I. 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% 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. 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 chapter, 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 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 algorithms schedule 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.

II. ENERGY AWARE 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 M and R be the set of map and reduce tasks of the application, and A and 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 eij the difference between energy consumption of slot $j \in \{A, B\}$ Bg when executing task $i \in \{M, R\}$ and its idle energy consumption. In addition, we denote by p_{ij} the processing times of task $i \in \{M, R\}$ when executed on slot $j \in \{A, B\}$. We assume that the processing time of the tasks is 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 $\delta_{ij}, \forall t, i \in M \cup R$ characterizing the dependencies of the map and reduce tasks as follows:

$$\delta_{ti} = \begin{cases} 1 & If\ task\ i\ should\ be\ assigned\ after\ task\ t \\ 0 & Otherwise \end{cases} \quad (5.1)$$

We formulate the Energy-Aware MapReduce Scheduling problem as an Integer Problem (called EMRS-IP) as follows:

$$Minimize \sum_{j \in \mathcal{A}} \sum_{i \in \mathcal{M}} e_{ij} X_{ij} + \sum_{j \in \mathcal{B}} \sum_{i \in \mathcal{R}} \sum_{t \in \mathcal{M} \cup \mathcal{R}} \delta_{ti} e_{ij} Y_{ij} \quad (5.2)$$

Subject to

$$\sum_{j \in A} X_{ij} = 1, \forall i \in \mathcal{M} \quad (5.3)$$

$$\sum_{j \in \mathcal{B}} \sum_{t \in \mathcal{M} \cup \mathcal{R}} \delta_{ti} Y_{ij} = 1, \forall i \in \mathcal{R} \quad (5.4)$$

$$\sum_{j \in \mathcal{M}} p_{ij} X_{ij} + \sum_{i \in \mathcal{R}} \sum_{t \in \mathcal{M} \cup \mathcal{R}} \delta_{ti} p_{ij'} Y_{ij'} \leq D \quad \forall j \in \mathcal{A}, \forall j' \in \mathcal{B} \quad (5.5)$$

$$X_{ij} = \{0,1\}, \forall i \in \mathcal{M}, j \in A \quad (5.6)$$

$$Y_{ij} = \{0,1\}, \forall i \in \mathcal{R}, j \in \mathcal{B} \quad (5.7)$$

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

$$X_{ij} = \begin{cases} 1 & If\ map\ task\ i\ is\ assigned\ to\ slot\ j \\ 0 & Otherwise \end{cases} \quad (5.8)$$

$$Y_{ij} = \begin{cases} 1 & If\ reduce\ task\ i\ is\ assigned\ to\ slot\ j \\ 0 & Otherwise \end{cases} \quad (5.9)$$

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 (5.3) ensure that each map task is assigned to a slot for execution. Constraints (5.4) ensure that each reduce task is assigned to a slot. Constraints (5.5) ensure that processing time of the application does not exceed its deadline. Constraints (5.6) and (5.7) represent the integrality requirements for the decision variables. The solution to EMRS-IP consists of X and $\bar{Y}$ where

$\widehat{Y_{ij}} = \sum_{t \in \mathcal{M} \cup \mathcal{R}} \delta_{ij} Y_{ij}$, $i \in \mathcal{R}$,and j $\in \mathcal{B}$

Note that based on constraints (5.5), 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 big data 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 study.

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 sub problems 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.

III. ENERGY AWARE MAPREDUCE SCHEDULING ALGORITHM

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

EMRSAII, 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 the following algorithm.

The design of these algorithms requires 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:

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:

$$ecr_j^m = \min_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}, \forall j \in \mathcal{A} \quad (5.10)$$

$$ecr_j^r = \min_{\forall i \in R} \frac{e_{ij}}{p_{ij}}, \forall j \in \mathcal{B} \quad (5.11)$$

where ecr_j^m and ecr_j^r 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:

$$ecr_j^m = \frac{\sum_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}, \forall j \in \mathcal{A} \quad (5.12)$$

$$ecr_j^r = \frac{\sum_{\forall i \in R} \frac{e_{ij}}{p_{ij}}}{R}, \forall j \in \mathcal{B} \quad (5.13)$$

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_j^m means that slot j has a higher priority to have a map task assigned to it. Similarly, a lower ecr_j^r 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 reduces processing times, denoted by f, in order to balance the assignment of map and reduce tasks. The ratio f is defined as follows:

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

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 pijm and pijr) is used by EMRSA-X to compute the ratio f.

A key challenge when designing the algorithms is that the user only species 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 con gurations 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 Q^m and 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 reduces 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 reduces 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 reduces deadlines based on the allocated tasks (lines 19-21)

Algorithm 1- EMRSA – X

Step 1: Create an empty priority queue Q^m
Step 2: Create an empty priority queue Q^r
Step 3: for all $j \in \mathcal{A}$ do
Step 4: $ecr_j^m = \min_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}$ for EMRSA -1 or

$ecr_j^m = \frac{\sum_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}$ for EMRSA – 2
Step 5: $Q^m.enqueue\ (j, ecr_j^m)$
Step 6: for all $j \in \mathcal{B}$ do
Step 7: $ecr_j^r = \min_{\forall i \in R} \frac{e_{ij}}{p_{ij}}$ for EMRSA -1 or

$ecr_j^r = \frac{\sum_{\forall i \in R} \frac{e_{ij}}{p_{ij}}}{R}$ for EMRSA – 2
Step 8: $Q^r.enqueue\ (j, ecr_j^r)$
Step 9: $D^m \leftarrow \infty\ ;\ D^r \leftarrow \infty$
Step 10: while Q^m is not empty and Q^r is not empty do
Step 11: $j^m = Q^m.extractMin()$
Step 12: $j^r = Q^r.extractMin()$
Step 13: $f = \frac{\sum_{\forall i \in \mathcal{M}} p_{ij}^m}{\sum_{\forall i \in R} p_{ij}^r}$
Step 14: $\mathcal{T}^m$: sorted unassigned map tasks $i \in \mathcal{M}$ based on p_{ij}^m
Step 15: $\mathcal{T}^r$: sorted unassigned reduce tasks $i \in R$ based on p_{ij}^r
Step 16: if $\mathcal{T}^m = \emptyset$ and $\mathcal{T}^r = \emptyset$ then break
Step 17: ASSIGN-LARGE ()
Step 18: ASSIGN-SMALL ()
Step 19: if $D^m = \infty$ then
Step 20: $D^m = D - p^r$
Step 21: $D^m = p^r$
Step 22: if $\mathcal{T}^m \neq \emptyset$ and $\mathcal{T}^r \neq \emptyset$ then
Step 23: no feasible schedule
Step 24: Output: X,Y

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 im and reduce task ir from the sorted sets T m and T r, respectively (lines 1- 2). Then it checks the feasibility of allocating map task im to slot jm and reduce task ir to slot jrby

checking the total processing time of the tasks against the deadline D (line 4).

Algorithm 2: ASSIGN-LARGE()

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

If the assignment of map task im and reduce task ir is feasible, the algorithm continues to select tasks from T^m and 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 (lines 22).

Algorithm 3: ASSIGN-SMALL()

Step 1: {Assign small map tasks}
Step 2: $i = argmin_{t \in \mathcal{T}} p_{tj}^m$
Step 3: while $p^m + p^r + p_i^m j^m \leq D$ and $p^m + p_i^m j^m \leq D^m$ and $\mathcal{T}^m \neq \emptyset$ do
Step 4: $\mathcal{T}^m = \mathcal{T}^m \setminus \{i\}$
Step 5: $p^m = p^m + p_i^m j^m$
Step 6: $X_{ij^m} = 1$
Step 7: $i = argmin_{t \in R} p_{tj}^r$
Step 8: $\mathcal{T}^r = \mathcal{T}^r \setminus \{i\}$
Step 9: $p^r = p^r + p_i^r j^r$
Step 10: $Y_{ij^r} = 1$
Step 11: $:i = argmin_{t \in \mathcal{T}^r} p_{tj}^r$

After allocating the map and reduce tasks with the largest processing time, EMRSAX assigns small map and reduce tasks while satisfying the deadline by calling ASSIGNSMALL() (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).

The time complexity of EMRSA-X is O(A(M+logA)+B(R+logB)+min(A;B)(M logM+ RlogR)), 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.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

A. 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 [64]. 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 1. The cluster is composed of four Intel nodes, with one node as a master. Two of the nodes have 24GB memory, 16 2.4GHz Intel processors, and a 1TB Hard Drive. The other two nodes have 16GB memory, 16 2.4GHz Intel processors, and a 1TB Hard Drive. The cluster has a total of 80GB memory, 64 processors, 4TB 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 1800 Watts. The measurement accuracy is +/- 1.5% 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 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 1500 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. 6.1 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 2.93GHz hexa-core dual-processor systems with 90GB of RAM which are part of the Wayne State Grid System.

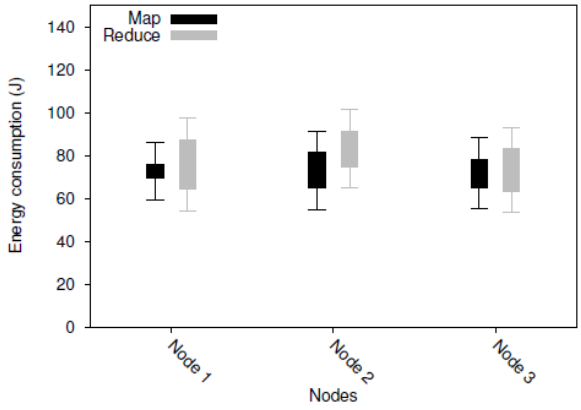

Figure 1: Energy needs of tasks for the actual and simulated architectures

B. Result Analysis and Discussions

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 1. For example, the smallest job represented by (48M; 48R) has 48 map tasks and 48 reduce tasks. Figure 2 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% 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 5356, 5396, 5233, and 8687 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% 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 1: 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

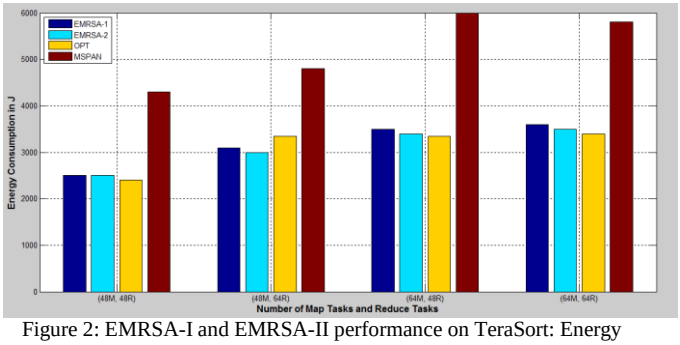

Figure 2: EMRSA-I and EMRSA-II performance on TeraSort: Energy consumption

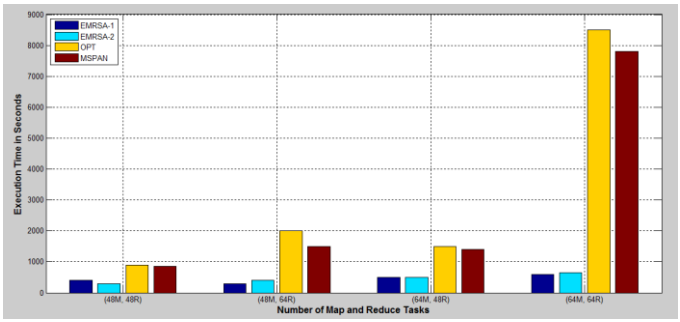

Figure 3: EMRSA-I and EMRSA-II performance on TeraSort: Execution time

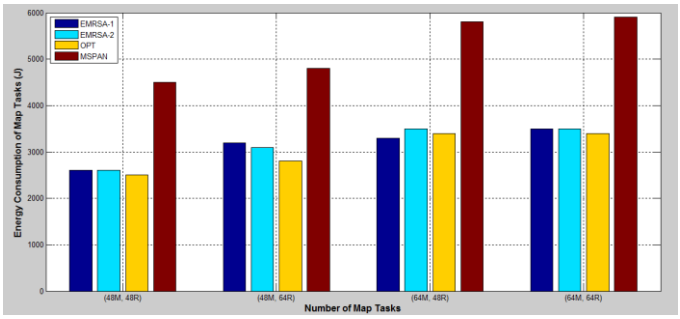

Figure 4: TeraSort energy consumption (small-scale experiments): Map tasks

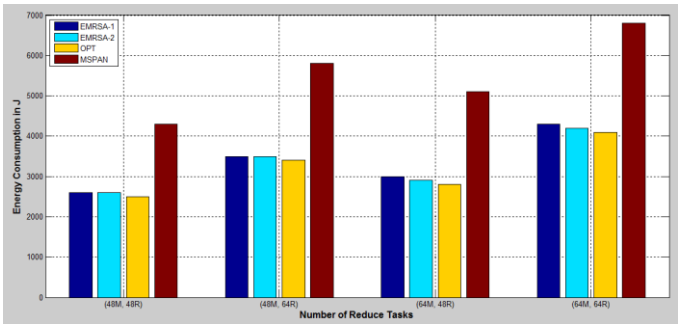

Figure 5: TeraSort energy consumption (small-scale experiments): Reduce tasks

Figure 3 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.

In Figure 4 and Figure 5, 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, EMRSAII, OPT, and MSPAN are 3130, 3090, 2897, and 4751 J, respectively, while the energy consumptions of reduce tasks for workload (48M; 64R) are 3547, 3527, 3448, and 5972 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 Figure 4) is closer to the optimal than the energy consumption for the reduce tasks (shown in Figure 5) 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.

V. CONCLUSION

Although cloud computing has gained a great deal of attention, there are still some key impediments to large scale enterprise adoption. The ever-growing demand for cloud resources from businesses and individuals places the cloud resource management at the heart of the cloud providers' decision-making process. One of the major challenges faced by the cloud providers is cloud resource management. Cloud providers and cloud users, pursue different goals. Cloud providers aim to maximize revenue while achieving high resource utilization. On the other hand, users want to minimize their expenses while meeting their performance requirements. However, the challenge is how to allocate and price resources in a mutually optimal way despite the lack of information sharing among users and cloud providers. In addition, the cloud environment is highly variable and unpredictable. Cloud providers may oversubscribe users to a shared infrastructure to increase their resource utilization, while oversubscription will results in resource contention and interference. In addition, there are other factors that contribute to unpredictability of the environment such as heterogeneity of the VMs. These factors make these multi-criteria optimization problems very complex.

One of the main concerns for a cloud service provider is minimizing the operational cost, especially the cost of power consumption. It is possible to reduce power consumption of the hardware by means of deploying more Virtual Machines (VMs) onto fewer hosts. This mechanism will provide the cloud providers the flexibility of dynamically determining the price of their resources and their cost share. The cloud providers will be free from building complex pricing models or generating user statistics for prediction of system usage. In addition, the cloud providers can use our proposed energy-aware schedulers in their data centers to lower their energy costs leading to lower service prices offered to their users. On the other hand, different types of users will be able to select their desired usage of cloud services.

REFERENCES

[1] J. Koomey. Growth in data center electricity use 2005 to 2010. Oakland, CA: Ana-lytics Press. August, 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: Simpli ed data processing on large clusters. In Proc. of the 6th USENIX Symp. on Operating System Design and Implementation, pages 137-150, 2004.

[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. Technical Report UCB/EECS-2009-55, UC Berkeley, 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. Ayguade. Resource-aware adaptive scheduling for mapreduce clusters. In Proc. of the 12th ACM/IFIP/USENIX Intl. Middleware Conf., pages 187-207, 2011.

[8] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: automatic resource inference and allocation for mapreduce environments. In Proc. 8th ACM Int'l Conf. on Auto-nomic Comp., pages 235-244, 2011.

[9] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: locality-aware resource alloca-tion for mapreduce in a cloud. In Proc. Conf. High Performance Comp., Networking, Storage and Analysis, 2011.

[10] 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. 20th IEEE Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecom. Syst., pages 11-18, 2012.

[11] H. Chang, M. S. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukher-jee. Scheduling in mapreduce-like systems for fast completion time. In Proc. of the 30th IEEE Intl. Conf. on Computer Communications, pages 3074-3082, 2011.

[12] F. Chen, M. S. Kodialam, and T. V. Lakshman. Joint scheduling of processing and shu e phases in mapreduce systems. In Proc. of the IEEE INFOCOM, pages 1143- 1151, 2012.

[13] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlos. On scheduling in map-reduce and ow-shops. In Proc. 23rd Annual ACM Symp. on Parallelism in Algorithms and Architectures, pages 289-298, 2011.

[14] Y. Zheng, N. B. Shro , and P. Sinha. A new analytical technique for designing provably efficient mapreduce schedulers. In Proc. of the IEEE INFOCOM, pages 1600-1608, 2013.