Not All Joules are Equal: Towards Energy-Efficient and Green-Aware Data Processing Frameworks

Zhaojie Niu, Bingsheng He
Nanyang Technological University, Singapore

Fangming Liu
Huazhong University of Science and Technology, China

***Abstract*—Interests have been growing in integrating renewable energy into data centers, which attracts many research efforts in developing *green-aware* algorithms and systems. However, little attention was paid to the efficiency of each joule consumed by data center workloads. In fact, not all joules are equal in the sense that the amount of work that can be done by a joule can vary significantly in data centers. Ignoring this fact leads to significant energy waste (by 25% of the total energy consumption in Hadoop YARN on a Facebook production trace according to our study). In this paper, we investigate how to exploit such *joule efficiency* to maximize the benefits of renewable energy for MapReduce framework. We develop job/task scheduling algorithms with a particular focus on the factors on joule efficiency in the data center, including the energy efficiency of MapReduce workloads, renewable energy supply and the battery usage. We further develop a simple yet effective performance-energy consumption model to guide our scheduling decisions. We have implemented GreenMR, an energy-efficient and green-aware MapReduce framework, on top of Hadoop YARN. The experiments demonstrate the accuracy of our models, and the effectiveness of our energy-efficient and green-aware optimizations over Hadoop YARN and a state-of-the-art green-aware Hadoop YARN implementation.**

I. Introduction

As emerging computing infrastructures, data centers consume up to 3% of all global electricity production while producing 200 million metric tons of CO2 in 2014 [1]. In order to reduce carbon emissions and financial burden on the electricity, many data centers have been built, being powered at least partially by renewable energy (or green energy, such as solar and wind). On one hand, data centers have been re-designed and integrated with the intelligence of smartly drawing power from multiple sources, including *green* energy from renewable and non-polluting sources and *brown* energy from traditional polluting sources [2]. For example, the renewable energy is utilized when it is available and the brown energy is drawn from electric grid when renewable energy is insufficient for powering the data center. Reducing the amount of brown energy consumption is the key for reducing the environmental pollution and can also reduce the electricity bills [3]. Under this context, green-aware systems that integrate the awareness of renewable energy has become a hot research topic (e.g., [4], [5], [3], [6]). On the other hand, data processing frameworks (such as MapReduce [7]) are becoming more and more important workloads in data centers, which contribute to significant portions of energy consumption. Particularly, we consider each job can be delayed by a bounded amount of time and we have the opportunity to schedule job/task for utilizing green energy. Overall, this paper investigates whether and how we can reduce the brown energy consumption for data processing frameworks in the data center with both brown and renewable energy supply.

The key challenge of exploiting renewable energy is that the sources are intermittent due to daily/seasonal effects. Thus, the renewable energy supply may not match the workload demand, which results in the severe under-utilization of renewable energy in a non-green-aware system. Green-aware algorithms and systems (e.g., [3], [6], [4]) have been developed to address the mismatch in the context of data centers. The core ideas behind those studies are similar: they delay workloads according to jobs' deadline to match the renewable supply.

While those green-aware algorithms and systems can exploit the green energy *at a coarse-grained level*, they fail to capture the efficiency of each joule. We define the concept of *joule efficiency* to be the amount of work that can be done by a joule. Not all joules are equal in the sense that the joule efficiency of the energy can vary significantly, depending on when the energy comes and how the energy is used in computing. Ignoring joule efficiency can lead to significant energy waste (by 25% of the total energy consumption of Hadoop YARN on a Facebook production trace according to our study in Section IV).

We have observed a number of key factors resulting in joule efficiency of data processing frameworks in data centers. First, data processing frameworks, particularly MapReduce, has non-linear performance speedup feature when the number of machines increases. We run the GridMix benchmark and the detail of the experimental setup can be found in Section IV. We can see that running on fewer nodes is actually more energy efficient (the total energy consumed for the job is lower) than running on more nodes. Therefore, one joule of energy can result in different amounts of work done, depending on the workload characteristics and the number of machines involved in the job. The largest difference is around 35%, which shows the significant energy waste if the energy efficiency is ignored. Second, the renewable energy supply is intermittent and time-varying, which may not match well with the most energy-efficient execution plan of the MapReduce workload. Third, battery has become an integral component for many data centers in order to guarantee the availability of the data centers [8]. Charging/discharging causes inherent energy loss. Ideally, a green-aware system should be aware of not only the green energy supply, but also joule efficiency to maximize the effectiveness of utilizing green energy.

In this paper, we propose *GreenMR*, an energy-efficient and green-aware MapReduce framework. With the slack time on each job, we develop job/task scheduling algorithms by

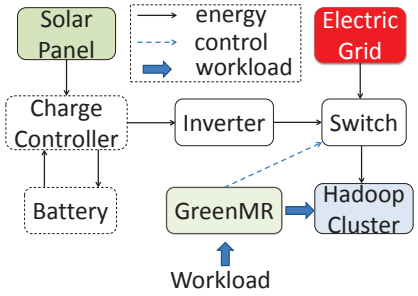


Figure 1: System overview of GreenMR

special considerations on the factors on joule efficiency in the data center, including the energy efficiency of MapReduce workloads, renewable energy supply and the battery usage. Since finding the optimal scheduling solution is a NP-hard problem, GreenMR embraces a series of simple and effective heuristics for optimizations. We further develop a performance-energy consumption model. The model guides us to make the scheduling decision so that the deadline can be met and joule efficiency can be accurately estimated for scheduling the job/task at appropriate time.

We have implemented GreenMR on top of the Hadoop YARN (2.4.0). We evaluate GreenMR in two complementary approaches. First, we run real experiments with GreenMR on a local cluster with micro benchmarks. Second, to allow more comprehensive studies, we have developed a simulator that replays the traces from the Hadoop cluster. Our experimental results demonstrate that 1) Our performance-energy consumption model can accurately predict the performance and energy consumption of our local cluster, 2) GreenMR significantly reduces the brown energy on both real experiments and simulations (up to 35% and 28% reduction compared with Hadoop YARN and GreenHadoop (the state-of-the-art green-aware Hadoop) [4], respectively). Moreover, GreenMR has a lower total energy consumption, by 21% and 19% smaller than Hadoop YARN and GreenHadoop, respectively.

The remainder of this paper is organized as follows. Section II formally defines the problem. Section III presents our detailed design of GreenMR, followed by the experiment results in Section IV. We review the related work in Section V and conclude this paper in Section VI.

II. Problem Definition

In this paper, we consider reducing the brown energy usage of data processing frameworks (particularly MapReduce) in a single data center. The data center is powered by both brown energy and green energy sources, with battery supports.

GreenMR allows users to specify the slack of a job: each job has a specified deadline to define its slack. The deadline is "soft", mainly representing the quality of service.

The system overview of GreenMR is illustrated in Figure 1. The system comprises of a Hadoop cluster, a charge controller, an inverter, batteries and a switch. As the input voltage from the renewable energy source varies dynamically, the charge controller is used to prevent any overcharging. The charge controller also monitors the charging/discharging operations. With the charge controller, surplus green energy is automatically charged into battery.

The data center has a switch connected with both green sources and brown sources (e.g., public grids). Research has been devoted to improve the effectiveness of this kind of switch [6]. When the power demand is higher than green power supply and/or the battery supply, it immediately draws power from brown sources.

Data centers may have centralized batteries to provide an uninterrupted power supply (UPS) for power failures. More recently, data centers also incorporate batteries at the rack- or even server-level, with higher energy efficiency than centralized battery. Previous studies [9] have demonstrated that batteries can be used to reduce the energy consumption cost in the scale of data centers. Without loss of generality, we model the battery with two parameters $< C_p, e >$, where C_p is the effective capacity of the battery that can be used for discharging and e is the charging efficiency of the battery ($0 \leq e \leq 1$). For a given charging power Δx, we estimate the energy storage increment as $\Delta x' = \Delta x \cdot e$. Here, we assume the charging efficiency as constant for simplicity. GreenMR can be extended to handle other battery models with variable efficiency. Due to the inherent charging/discharging loss, the battery is either charged or discharged at each time.

Optimization goal. Given the MapReduce workload with predefined slacks, GreenMR dynamically schedules the workloads according to the green energy supply and joule efficiency. The optimization goal is to minimize the total amount of brown energy usage, given the constraint that all workloads are completed before their predefined deadlines.

III. Design and Implementation

In this section, we start with an overall design of GreenMR. Then, we give detailed designs of its key components.

A. Overall Design

We propose a MapReduce framework aware of joule efficiency (GreenMR), and develop effective cost models and efficient optimization algorithms to address the above-mentioned technical issues. The overall design of GreenMR is described in Algorithm 1. When a new job comes to the system, it is initially put into a job waiting queue. For the jobs in the job waiting queue, we need to conduct the performance and energy consumption estimations on the job (Line 2). With these estimations, we determine the most energy-efficient execution configuration that satisfies the deadline (Line 3). The configuration includes the number of servers and the amount of physical resources per server. Then, GreenMR adds the job to the job queue (Line 4). The job queue of GreenMR is a *multi-queue* structure to express the energy-efficient execution plan.

Algorithm 1 Overall design of GreenMR

1: **if** A new job J comes **then**
2: Conduct the performance and energy consumption estimations on J;
3: Determine the execution configuration that has the smallest energy consumption and satisfies the deadline;
4: Add J and its energy-efficient configuration to the job waiting queue;
5: **if** Current time is the beginning of an epoch **then**
6: Conduct predictions on the workload and the green energy supply;
7: Generate the basic energy-efficient plan, P;
8: Optimize P by considering job transformations;
9: Optimize P with battery assisted green shifting;

GreenMR periodically generates the energy-efficient and green-aware scheduling plan (Lines 5-9). At the beginning of each epoch, we perform predictions on workloads and the green supply. Given the most energy-efficient execution configuration of each job, we first generate the basic energy-efficient plan according to the cluster capacity. Next, we apply a series of optimizations to reduce the brown energy consumption of the plan, including job transformations and battery assisted green shifting.

GreenMR periodically schedules jobs/tasks and performs power management on servers according to the optimized execution plan (the period is called a *slot*). In our design, we consider fine-grained jobs/tasks scheduling and power management and an epoch consists of multiple slots. For each time slot, GreenMR decides the set of active servers and allocates them to jobs and executes them according to the execution plan. The other servers are transited to low-power idle state (e.g., ACPIs S3). For all jobs to be scheduled in the current slot, GreenMR adopts the deadline first scheduling algorithm, which gives priority to the jobs close to deadlines.

Finally, we note that the epoch/slot design improves the robustness of GreenMR to inaccurate energy usage predictions or other dynamic factors including hardware failure, I/O contention and workload imbalance [10]. GreenMR re-evaluates and reacts to the system state every epoch/slot. For example, if the current epoch is over-estimating the energy usage, it may be forced to use more brown energy when not necessary. However, scheduled jobs can finish faster than expected, causing GreenMR to adjust by scheduling more jobs from the job queue. In the following subsections, we present the details on each key component in GreenMR.

B. Execution Plan Optimizations

We generate a basic execution plan with special considerations on energy efficiency of data processing frameworks and cluster capacity. Next, we further reduce the brown energy consumption of the basic energy-efficient plan by job transformations and battery assisted green shifting mechanisms.

1) Basic Energy-efficient Plan: At the beginning of an epoch, we perform online predictions on the workload (including job types and arrival rates etc) and on the green supply for the epoch.

Workload prediction. Generally, it is difficult to have an accurate prediction on the workload, because of the diversifying job sizes [11]. Fortunately, we observe that small jobs and large jobs in real production traces have very different arrival patterns (see the experimental setup in Section IV-A). Particularly, the arrival rate of small jobs (# maps is less than ten in our experiment) can be predicted with relatively high accuracy by using some simple time series analysis [12] (prediction error is less than 8% in average for the real production trace from Facebook). In our experiment, we apply exponentially weighted moving average algorithm based on the historical statistics data to predict the arrival rate for small jobs. The total energy consumption and execution time of these small jobs can be estimated accurately with similar time series prediction algorithms. In our experiments, the prediction error of energy consumption is less than 6% in average and prediction error of execution time is less than 4% in average.

As for large jobs, they come in a more ad hoc manner, and the energy consumption and execution time of large jobs vary significantly. Instead, we estimate large jobs individually using our cost model after they come. Since large jobs tend to have longer execution time, such an on-demand prediction has given sufficient optimization room for scheduling, as we observed in our experiments.

Green supply prediction. Most green sources such as solar and wind depend on weather. There are a number of existing weather learning methods and models (e.g., [4], [13]). This is not the focus of this paper. We simply adopt the previous approach used in GreenHadoop [4], which has been shown to achieve very good accuracy for solar energy. Additionally, we also explicitly study the impact of prediction errors in the experiments.

Execution plan generation. We now generate the basic energy-efficient plan for GreenMR. We note that, different from the SDE plan generated in GreenHadoop [4], the plan generated by GreenMR has two distinct features. First, our job execution configuration has the smallest energy consumption while satisfying the deadline. Second, the plan generation is guided by our energy consumption and performance model for MapReduce jobs, rather than heuristics [4].

We develop a multi-queue to express the execution plan of an epoch. Each epoch consists of N time slots and the slot length is t_s seconds, where one time slot corresponds to one queue. There are N queues for the current epoch in our design, expressed with $q_0, q_1, \ldots, q_{N-1}$, where $q_i (0 \leq i < N)$ maintains all jobs with the slack time less than $(i+1) \cdot t_s$. A separate queue q' is used to store all jobs with the slacks beyond the current epoch. Suppose a job with a slack of k is submitted after t seconds of the beginning of current epoch, it will be added to $q_{\frac{t+k}{t_s}}$ if $t+k$ is within the current epoch, otherwise it will be added to waiting queue q'. As the time goes by, at the beginning of each time slot, the slacks of jobs belongs to the current queue will decreases to $[0, t_s]$, and they all will be assigned with the decided number of servers which are scheduled to run until all tasks completes. At the beginning of each epoch, jobs with the slacks smaller than the end of the epoch are distributed to corresponding q_i, and jobs have not finished in the previous epoches will be distributed to q_0. Thus, the jobs for an epoch include the workload from q' in the previous epoch and the jobs that arrive during the current epoch.

After initiating each job into the corresponding queue, we ensure each slot can schedule all jobs/tasks allocated to it subject to the cluster capacity. Suppose the numbers of servers that are required in each time slot are $n_i (0 \leq i < N)$. If n_i exceeds the cluster capacity, we need to move the jobs running in current time slot with the earliest submission time to the queue j ($j < i$) with available resources. We repeat this process until all n_i are no larger than the cluster's capacity. Otherwise, the cluster is overloaded.

2) Optimizations with Job Transformations: We first define four basic job transformation operations, and next present the optimization techniques that leverage these defined op-

erations.

GreenMR supports four basic job transformation operations: 1) $Delay(j,i)$: it delays the starting time of job j for i time slots; 2) $Advance(j,i)$: it advances the starting time of job j for i time slots; 3) $Promote(j,i)$: it increases the number of machines allocated to the job j by one in time slot i, so that it can complete faster but usually at the cost of more energy consumption; and 4) $Demote(j,i)$: it reduces the number of machines allocated to the job j by one in time slot i, which is a dual operation of promotion. Promotions and demotions capture the non-linear performance and energy consumption in MapReduce jobs. Our cost model can predict the performance and energy consumption changes for the job transformations.

Those four operations can be used together to form a sequence of transformation operations. Additionally, the same operations can be used multiple times. For example, we should use promotions for x times if we want to add x machines to the job.

Given the generated basic energy-efficient plan of the current epoch and the predicted green energy distribution in each time slot, we need to perform the transformations for each job using the above four operations, and our goal is to minimize the total brown energy consumption in the current epoch among all jobs. We need to decide how many machines to be assigned in each time slot for every job under the deadline and resource capacity constraints. We can formulate this problem into a temporal knapsack problem [14]. We omit the formulation due to the space limitation. Since finding the optimal solution to that problem is NP-hard, we propose a series of simple and effective heuristics for optimizations. Particularly, we propose a novel two-phase heuristics-based approach to solve this non-trivial optimization problem. Those two phases are complementary with each other. Phase 1 algorithm utilizes the delay and advance operations to shift the basic energy-efficient plan as a whole without impact on its energy efficiency. Phase 2 algorithm leverages the demote and promote operations to change the shape of the basic energy-efficient plan at the cost of the energy efficiency.

Phase 1: In this phase, we perform advance and delay operations to reduce the brown energy consumption while preserving the joule efficiency of the basic energy-efficient execution plan of the current epoch. As all jobs in the basic energy-efficient execution plan are delayed as late as possible, any delay operations will cause deadline violations. Therefore, we first perform the advance operations and then consider the delay operations if the brown energy consumption can be further optimized.

The optimization algorithm for Phase 1 is described in Algorithm 2. First, we perform advance operations on the basic energy-efficient execution plan. We iterate the execution plan in ascending order (from the first time slot to the last time slot in the epoch). For each time slot, we consider all jobs that are submitted before that time slot while start after that. If the brown energy consumption of the current epoch can be reduced by advancing the starting time of any job, we perform the advance operation on that job. Note, the brown energy consumption of an epoch is estimated to be $\sum_{i=1}^{N} Max\{J_i - G_i, 0\}$, where N is the number of time slots in one epoch, J_i is the total energy consumption of all jobs running in time slot i (estimated using the cost model), and G_i is the predicted amount of green energy in time slot i. Then, we perform delay operations on the basic energy-efficient execution plan. We iterate the execution plan in reverse order (from the last slot forwarding to the first slot in the epoch). For each time slot, we consider all jobs start before that time slot. We iterate these jobs and perform delay operations on them if the brown energy consumption can be reduced.

Algorithm 2 Optimization algorithm of Phase 1

```
1: for each time slot t of current epoch in ascending order do
2:     J = jobs submitted before time t and start after time t;
3:     for each job j in J do
4:         u = brown energy consumption of the current execution plan;
5:         s = start time of job j;
6:         o = advance the start time of job j for s − t time slots;
7:         if o can be performed on the execution plan then
8:             v = brown energy consumption after performing operation o;
9:             if v ⩽ u then
10:                Advance(j, s − t);
11: for each time slot t of current epoch in reverse order do
12:    J = jobs start before time t;
13:    for each job j in J do
14:        u = brown energy consumption of the current execution plan;
15:        s = start time of job j;
16:        o = delay the starting time of job j for t − s time slots;
17:        if o can be performed on the execution time then
18:            v = brown energy consumption after performing operation o;
19:            if v ⩽ u then
20:                Delay(j, t − s);
```

Phase 2: Complementary to Phase 1, we consider promote and demote operations on the jobs if those transformations can further reduce the brown energy consumption. Instead shifting an execution plan of a job as a whole, Phase 2 algorithm can change the shape of the execution plan at the cost of energy efficiency.

The optimization algorithm for Phase 2 is described in Algorithm 3. We iterate the execution plan optimized after Phase 1 in ascending order. For each time slot t in the current epoch, we consider workload shifting through promote and demote operations to further reduce the brown energy consumption. We add all jobs, which will be scheduled in the current time slot, into a FIFO queue. We dequeue a job from the FIFO queue each time and relocate the machine assigned to the job in the current time slot to the slot with the most surplus green energy supply if the brown energy consumption can be reduced. The tasks to be scheduled on this machine are also shifted accordingly. After the shifting, we enqueue the current job in to the FIFO queue and repeats this process until the FIFO queue becomes empty. We choose the FIFO queue structure to make all jobs benefit from the green energy evenly, which also minimizes the total reduction of joule efficiency caused by the non-linear speedup feature of Hadoop.

3) Battery Assisted Green Shifting: We can take advantage of battery to store the green energy for future use when the green energy supply is surplus. The (**basic** approach) adopted by the previous studies (e.g., [3], [5]) are as follows. The battery is charged only when there is surplus green energy (i.e., the green energy supply is higher than the total energy consumption). When the energy demand is higher than green supply, the battery is first discharged. The discharging stops when the battery capacity reaches a predefined threshold, only

Algorithm 3 Optimization algorithm of Phase 2

```
1: for each time slot m in the current epoch do
2:     Q = ∅; /* FIFO queue */
3:     for each job j in time slot m do
4:         Q.enqueue(j);
5:     while Q ≠ ∅ do
6:         u = brown energy consumption of the current execution plan;
7:         j = Q.dequeue();
8:         for each time slot n in slots with surplus green energy do
9:             o = demote job j at time m and promote job j at time n;
10:            if o can be performed on the execution plan then
11:                v = brown energy consumption after performing operation o;
12:                if v < u then
13:                    Promote(j, n);
14:                    Demote(j, m);
15:                    shift corresponding tasks from slot m to n;
16:                    u = v;
17:                    Q.enqueue(j);
18:                    break;
```

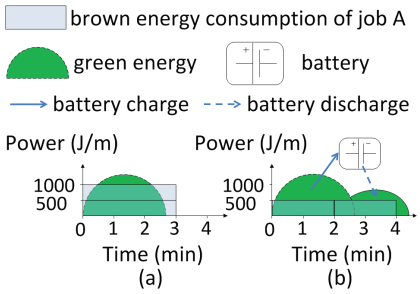


Figure 2: An example of battery assisted green shifting

for emergency use (e.g., when external energy sources are unavailable), and the brown energy is used.

However, the basic approach does not consider *joule efficiency*. If we find that storing the green energy can do more work in the future (green shifting) rather than using it at present, we can proactively store it into the battery despite of the charging loss. This charging method can be feasible, since MapReduce jobs exhibit non-linear speedup and energy consumption behavior.

With the consideration of joule efficiency, the charging of battery needs careful designs. Figure 2 gives an example to show that how to reduce the brown energy consumption through green energy shifting with the support of battery. In Figure 2 (a), all green energy is utilized directly. Instead, Figure 2 (b), only partial green energy is used after a job transformation (demotion) and remainder is charged to the battery for future use. It results in significant brown reduction as green energy is utilized more efficiently.

We develop our battery assisted green shifting as a further optimization for the execution plan optimized by job transformations. Our optimization is periodically performed at the beginning of each time slot. With the battery, we can delay workload to later slots for higher energy efficiency. As there is energy loss in charging/discharging, we need to assess the tradeoff between energy gain of battery assisted green shifting and the charging/discharging loss. Still, we can formulate this problem as a nonlinear integer programming problem with constraints (e.g., battery capacity and deadlines), which is NP-hard. Therefore, we propose a heuristic algorithm to perform the green shifting optimization on the execution plan after job transformations.

Green shifting algorithm: We iterate the execution plan in the ascending order. For each time slot, we consider performing demote and charging surplus green energy for future usage for all jobs in the current time slot. Similar to Algorithm 2, we firstly add all jobs J to be scheduled in the current time slot into a FIFO queue Q, and then dequeue a job from Q each time and perform demote as well as charging for future usage if brown consumption can be reduced. For each job j dequeued from Q, if demotion will not cause deadline violation, we perform demote operation on job j. Demote operation on j causes less workload to be scheduled as surplus green is not fully utilized, The surplus green energy can be charged to battery and used to schedule the left workload for higher energy efficiency in the later time slots. As there is energy loss during charging/discharging, we need to evaluate the favor (energy saving) coming from the efficient energy usage and the energy loss before performing the green energy shifting. We only conduct the green energy shifting when the brown energy consumption can be reduced. Among all future time slots, the one which can gain the most brown energy reduction is chosen. After the shifting, we enqueue the current job in to the FIFO queue and repeats this process until the FIFO queue becomes empty. Similar to the Phase 2 algorithm, we choose the FIFO queue structure to minimize the impact on the reduction of the joule efficiency caused by the non-linear speedup feature of Hadoop.

C. Cost Model

Since we use the performance-energy consumption models at run time, we aim to develop a lightweight yet sufficiently accurate cost model. Since small jobs (i.e., with less than 10 map tasks) are predicted with time series algorithms, the model applies to large jobs only.

We have a number of design considerations. First, we need to integrate the hardware profile including servers and network switches to quantitatively reflect the relationship between utilization and energy consumption. We assume that the cooling energy consumption is strongly correlated to the total energy consumption of servers and network switches, and exclude the cooling energy consumption in our estimation for simplicity. Second, we need to have the total execution time and energy consumption of a MapReduce job, for the effectiveness of any scheduling decision. Third, the runtime overhead of the cost model should be low. There are a number of performance models for MapReduce (e.g., [15], [16]). We extend the prediction model used in Bazaar [15] according to our design principles for guiding the scheduling decisions of GreenMR. There is a tradeoff between runtime overhead and model accuracy. Our design decision leans towards simplicity with sufficient accuracy.

Overall, our performance-energy consumption model estimates the energy consumption and execution time, given the program for a MapReduce job, the size of the input data, and a execution configuration as input. Table I lists the parameters used in the cost model. The parameters are on the hardware models in the cluster and on the MapReduce executions.

Job profiling: It is generally difficult to estimate the cost of user-defined functions directly. Similar to the previous studies [15], we profile the MapReduce program by executing it using a random sample of the input data (in our experiment, the number of map tasks is ten). The profiler determines the

Table I: Variables in the cost model

T_m	Execution time of the map phase (sec)
T_r	Execution time of the reduce phase (sec)
P_m	Power consumption of the map phase (w)
P_r	Power consumption of the reduce phase (w)
T_m^{io}	Waiting time of disk I/O in the map phase (sec)
T_m^p	Processing time of the map functions (sec)
T_s	Shuffle time of the reduce phase (sec)
T_r^{io}	Waiting time of disk I/O in the reduce phase (sec)
T_r^p	Processing time of the reduce functions (sec)
U_m	CPU utilization of nodes in the map phase
U_r	CPU utilization of nodes in the reduce phase
U_s^m	Switch utilization in the map phase
U_s^r	Switch utilization in the reduce phase
N	# of nodes
N_s	# of Switches
R	Physical resources per node (CPU, MEM)
R_m	Physical resources required by each map task (CPU, MEM)
R_r	Physical resources required by each reduce task (CPU, MEM)
$\|I\|$	The size of the input data (MB)
B_d	Disk bandwidth under MapReduce-like access patterns (MB/s)
B_n	Network bandwidth (MB/s)
S_m	Data selectivity of map tasks
S_r	Data selectivity of reduce tasks
B_m	Processing rate of user-defined map function (MB/s)
B_r	Processing rate of user-defined reduce function (MB/s)
h	A factor captures the extra disk I/O relative to the data read in per reduce task

execution time for each task and each phase, the amount of data consumed and generated by each task. All those statistics are gathered from the log files generated during execution, and can be used to determine the data selectivity of map/reduce function (S_m, S_r) and the processing rate of map/reduce function (B_m, B_r). For instance, the processing rate of the job's map function can be easily obtained as the ratio of the data consumed by individual map functions to the actual running time of map function, measured with Linux "time" command. We take the averages of the sample executions as our predictions.

Hardware models: The hardware components include network switches and servers. Total power consumption of hardware consists of two types of power: static and dynamic consumption. Static consumption (including storage architecture, power supply, etc.) can be measured easily by keeping hardware in an idle state. Here we mainly study the dynamic consumption. For the server, we use the model from the previous study [17]. The basic idea is to define a function $\mathfrak{f}(c)$ as the power of a server, where c is the CPU utilization. For network switch, we adopt the power consumption model from a previous study [18]. It is also a utilization based model. The power of the switch is defined as $\mathfrak{g}(c)$, where c is the switch utilization. For space interests, we omit the details on calibrating the model in our local cluster and refer readers for more details in the original papers [17], [18], [19].

We model I/O and CPU processing of a MapReduce job execution with the following two phases.

Map phase: We divide the map phase into two components: disk I/O and data processing.

For the disk I/O, each map task reads its input split from local disk (assuming the input is locally available) ,and then writes the intermediate data generated by map function to local disk. It is bounded by the effect of the disk bandwidth. Thus, we calculate the waiting time of disk I/O in the map phase (T_m^{io}) to be $\frac{|I|\times(1+S_m)}{N\times B_d}$.

For the data processing, each map task applies the user-defined map function on its input split. Thus, We calculate the processing time of the map functions (T_m^p) to be $\frac{|I|}{N\times(R/R_m)\times B_m}$.

We assume that the CPU is idle during the disk/network I/O, the CPU utilization of the nodes in map phase (U_m) is estimated to be $\frac{T_m^p}{T_m^{io}+T_m^p}$. The switch utilization in the map phase (U_s^m) is estimated to be zero. Hence, we calculate the execution time and power consumption of the map phase as follows,

$$T_m = T_m^{io} + T_m^p$$

$$P_m = N\times\mathfrak{f}(U_m) + N_s\times\mathfrak{g}(U_s^m)$$

Reduce phase: The reduce phase is divided into two components: data shuffling and reduce processing.

For the data shuffling, reduce tasks complete two operations. Each reduce task first reads its partition of the intermediate data across the network, and then merges and writes it to disk. Hence, the bandwidth of this operation is the minimal value of B_d and B_n, which is expressed as $Min(B_d,B_n)$. Next, the data is read from the disk and an external merge is performed before the data is consumed by the reduce phase. This operations is bounded by disk bandwidth B_d. We use a factor h to capture the extra disk I/O relative to the data read in for a reduce task, which is obtained from job profiling. Thus, we calculate the shuffle time (T_s) to be $\frac{|I|\times S_m}{N}\times\left\{\frac{1}{Min(B_n,B_d)}+\frac{h}{B_d}\right\}$.

For the reduce processing, each reduce task reads its input from disk, applies the user-defined reduce function and writes the final result to disk. Similar to the map phase, we calculate the waiting time of disk I/O in the reduce phase (T_r^{io}) to be $\frac{|I|\times S_m\times(1+S_r)}{N\times B_d}$ and the processing time of the reduce functions (T_r^p) to be $\frac{|I|\times S_m}{N\times(R/R_r)\times B_r}$.

The CPU utilization of the nodes in the reduce phase (U_r) is estimated to be $\frac{T_r^p}{T_s+T_r^{io}+T_r^p}$. Given the amount of data shuffled, we can easily derive the utilization U_s^r for the switches involved. Hence, we calculate the execution time and power consumption of the reduce phase as follows,

$$T_r = T_s + T_r^{io} + T_r^p$$

$$P_r = N\times\mathfrak{f}(U_r) + N_s\times\mathfrak{g}(U_s^r)$$

With these two phases modeled, the total execution time of the job is simply T_m+T_r, and the total energy consumed is $T_m\times P_m + T_r\times P_r$.

As we will see in the experiments, the cost model is simple and lightweight, with only a small number of pre-executed tasks and very low computation overhead. This profiling process is done for large jobs only. Also, the cost model is sufficiently accurate in capturing the performance and energy consumption of MapReduce/Hadoop, and is effective in guiding the scheduling and optimizations in GreenMR.

IV. Experimental Evaluations

In this section, we evaluate GreenMR on a local cluster and with simulations with production traces from Facebook.

A. Experiment Setup

We perform two complementary sets of experiments to evaluate GreenMR. The first set of experiments is on a real

deployment on a 10-node cluster using micro benchmarks including TeraSort in Hadoop APIs and GridMix[1] benchmarks. This small-scale experiment in the real cluster is to reveal micro-level details with full control of choices on scheduling optimizations and to evaluate the energy consumption and performance models. The second set of experiments is to perform simulations on the real-world trace from data centers (particularly from Facebook) to assess the effectiveness of GreenMR in large-scale systems.

In both sets of experiments in the local cluster and simulations, the battery and solar panels are simulated. The battery charging efficiency is set to be 0.82, according to our measurements on a Fujitsu lithium ion battery with 10.8 V voltage. By default, the battery capacities are 1 kWh and 20 kWh for the experiments in the local cluster and simulations, respectively. We use the real-world traces for solar energy from Measurement and Instrumentation Data Center (MIDC), because solar energy is widely available. By default, we use the two days (May 1-2, 2011). We consider different solar power scales to evaluate the impact of different numbers of solar panels. By default, the peak solar power supply is provisioned to the peak power usage of the cluster. The default time slot size is one minute and the epoch size is 20 minutes.

We implement GreenMR based on top of Hadoop YARN (2.4.0). We consider Hadoop YARN 2.4.0 as the baseline, denoted as "Hadoop". Hadoop uses the default scheduler, and jobs are scheduled for executions without delay. We choose the state-of-the-art green-aware Hadoop [4] ("GreenHadoop") for comparison. Unlike GreenMR, GreenHadoop has not considered the joule efficiency of MapReduce or the energy efficiency of the battery usage. We have got the original source code of GreenHadoop and adapted GreenHadoop to run on Hadoop YARN. To evaluate the separate benefits of individual optimization techniques, we manually enable/disable certain optimizations in both real deployment and simulations. Overall, we define three optimization variants: "GreenMR(BE)", "GreenMR(JT)" and "GreenMR". The detailed configurations of optimization techniques in all compared algorithms are summarized in Table II. GreenMR has all the optimizations on joule efficiency proposed in this paper. The difference between GreenMR(BE) and GreenMR(JT) measures the effectiveness of our job transformations. The difference between GreenMR(JT) and GreenMR measures the effectiveness of our battery assisted green shifting optimization. Also, all algorithms use battery, either in the basic approach or with our proposed battery assisted green shifting technique. Like GreenHadoop [4], all schedulers use the low power state (ACPI S3) for power saving, and use the same replication method for data availability when a machine is set to the low power state.

The runtime overhead of running GreenMR scheduler is small. In our experiment, that overhead for a typical MapReduce job is less than 0.1 millisecond on a single Intel Xeon X5675 CPU core. This is mostly ignorable for data-intensive workloads.

1) Micro benchmark setup: The local cluster consists of 10 nodes, each with two Intel Xeon X5675 CPUs (12 cores in total), 24 GB memory and 500G SATA disks. The idle and peak power consumptions of one machine are 150 w and 280 w, respectively. The machines are connected with 10Gb/sec Ethernet. One node is configured as master, and the others are deployed as slaves. We use multiple meters to measure the energy consumption of the whole system, including machines and networking.

Table II: Configuration of optimization techniques for all schedulers

Optimization / Scheduler	SDE	Energy-efficient Plan	Job Transformation	Basic Battery Usage	Green Shifting
Hadoop				✓	
GreenHadoop	✓			✓	
GreenMR(BE)		✓		✓	
GreenMR(JT)		✓	✓	✓	
GreenMR		✓	✓		✓

We assemble a micro benchmark according to the daily pattern in Google search [20]. We consider four kinds of jobs: Terasort and GridMix (WebdataSort, WebdataScan and APISort). By default, the input size of each job is around 80 GB and the deadline is one hour. We consider a mixed workload, where each job is randomly generated from four kinds of jobs.

We have run each experiment for five times. Variances among different runs are small, and we report averages.

2) Simulation setup: We implement a trace-driven simulator by taking as input the solar traces and workload traces from data centers. We simulate the charging/discharging life cycles of battery and the power-performance tradeoff of the workload. In particular, we use a synthetic trace that models multi-user production workload in Facebook[2] with a 600-node cluster. We mimic the characteristics of its jobs using "loadgen". Loadgen is a configurable MapReduce job from the Gridmix benchmark included in the Hadoop distribution. The deadline of each job is computed based on its arrival time, expected execution time and slack. *By default, the slack is set to 100% of the expected execution time.* This setting is reasonable in practice, in the sense that users can tolerate a longer slack for a larger job. We also experimentally study other slack settings. The servers is configured with the same performance-energy consumption models as those in our local cluster. We apply two-level tree-structured networks in our simulation: intra-pod switch with 1Gb/sec bandwidth, and inter-pod switch with 10Gb/sec bandwidth. Each pod has 32 machines.

We analyze the arrival pattern and energy consumption statistics of jobs with different sizes. Small jobs (# Maps is smaller than 10) contributes to over 70% of the number of jobs, but their total energy consumption is less than 6%. In contrast, the total energy consumption of the jobs with more than 1000 map tasks accounts for over 79% of the total energy consumption of the entire trace. As discussed in Section III-B, we develop a job size-aware prediction on performance and energy consumption. The arrival rate, energy consumption and execution time of small jobs is predicted with a time series algorithm, and large jobs are scheduled in an ad hoc manner.

[1]http://hadoop.apache.org/docs/stable/gridmix.html

[2]https://github.com/SWIMProjectUCB/SWIM/wiki

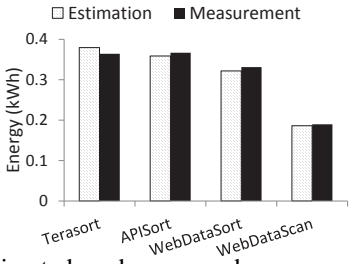


Figure 3: Estimated and measured energy consumption of individual jobs

B. Micro Benchmark Study

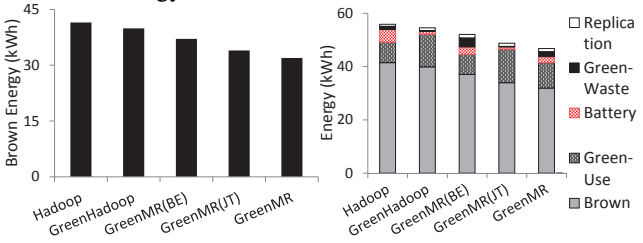
1) Model accuracy: We evaluate the accuracy of our cost model. We present the evaluation on the total energy consumption, since we have observed similar results on the performance. We evaluate the accuracy of our energy cost model by comparing energy consumption of individual jobs in our micro benchmark. We choose four kinds of jobs: Terasort and Gridmix (WebDataScan, WebDataSort and APISort). In our experiment, the inputs are generated uniformly by using TeraGen included in the Hadoop distribution and the data generators included in Gridmix benchmark. Figure 3 shows the estimated and measured energy consumption for running each job in the local cluster. For all jobs, the difference between estimation and measurement is smaller than 5% of the measurement when the number of profiling tasks is larger than 10, which validates the accuracy of the energy consumption model. We also evaluate the average prediction for Terasort and Gridmix workload by varying the number of map task used for profiling. As we can see from the Table III, our cost model achieves higher accuracy with the increase of map tasks to be profiled.

Table III: Prediction error with different profiling overhead

# map task used for profiling	5	10	20	40
average prediction error	12.8%	4.9%	3.5%	2.9%

2) Overall comparison: Figure 4(a) shows the brown energy consumption of different schedulers in running the micro benchmark. Overall, both GreenHadoop and GreenMR consume less brown energy than Hadoop, thanks to the alignment between power demand and green supply. Compared with GreenHadoop, GreenMR still consumes less brown energy, by embracing the energy-efficient optimizations. GreenMR(BE) reduces the brown consumption by 7.1% compared with GreenHadoop by considering energy efficiency. GreenMR(JT) applies job transformations to the basic energy-efficient plan and further reduces the brown consumption by 8.5% over GreenMR(BE). By integrating battery-assisted shifting, GreenMR reduces the brown consumption by 6% further over GreenMR(JT).

Figure 4(b) shows the energy breakdown. We divide the total energy into five categories: brown energy consumption, green energy consumed by the workload directly, green energy charged into battery, green energy wasted and data replication overhead. Green energy wasted includes energy loss during charging/discharging and the discarded part when battery is fully charged. GreenHadoop can utilize more green energy than Hadoop by aligning power demand with green supply. However, without considering joule efficiency, it uses more brown energy than GreenMR. Particularly, GreenMR(BE) utilizes every joule of energy more efficiently and consumes less brown energy than GreenHadoop. Job transformations in GreenMR(JT) use more green energy in an efficient manner. Finally, battery assisted green shifting techniques in GreenMR charges more green energy into battery rather than using it directly, because some green energy can be utilized with higher energy efficiency in the future and thus further reduces the brown energy.



(a) Brown energy consumption (b) Energy consumption breakdown

Figure 4: Overall comparison of the micro benchmark

We further study the replication overhead. Because we take advantage of ACPI S3 to put servers to a lower power state for energy saving, some data blocks should be replicated to other servers for processing. We observed that the replication overhead is negligible (smaller than 20% of the energy saving by transiting the server to S3 when appropriate). Two more issues are worth further discussions. First, the state transition of a server is lightweight. The period of a server transiting from S3 back to the active state is around 7 seconds, and transiting from the active state to S3 takes only 1 second. Second, the network performance of our cluster is reasonably good. A data transfer of one HDFS chunk of 128MB takes less than 0.3 second in our local cluster.

C. Large-scale trace study

We first evaluate the accuracy of our simulation, followed by the detailed results on the production trace.

1) Simulation validation: We first evaluate the accuracy of our simulation. We use the SWIM workload replay tool to scale down Facebook production trace for our local cluster. With a similar scale-down approach in the previous study [4], we scale down the workload as follows: we reduce the data by a factor of 8, and eliminate the largest 1.9% of jobs. We schedule the scaled-down workload in our local cluster and validate the simulation result with the real-measured value. Figure 5 shows the measured and simulated energy consumption. The difference between simulation and measurement is small (less than 2 kWh), which validates the accuracy of our simulator.

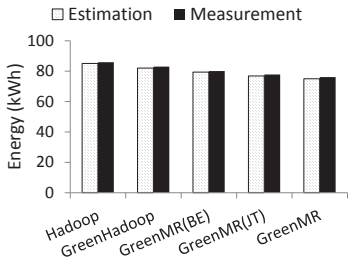
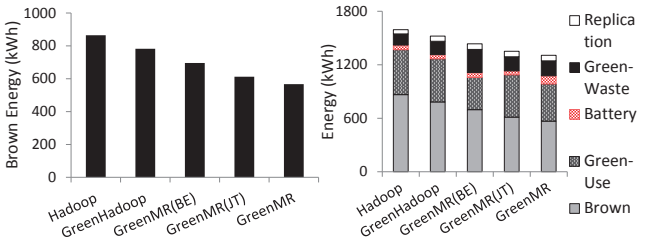


Figure 5: Simulation validation on replayed Facebook trace

2) Overall comparison: Figure 6(a) shows the brown energy consumption result on Facebook production workloads under five schedulers. The brown energy reduction of

GreenMR is even larger than that in the local cluster, by 35% and 28% over Hadoop and GreenHadoop, respectively. Moreover, GreenMR has a lower total energy consumption, by 21% and 19% smaller than Hadoop and GreenHadoop, respectively. That shows the importance of integrating energy efficiency into the system.

The energy consumption breakdown is shown in Figure 6(b). In the experiments, we have observed similar results to the local cluster. We highlight with the following observations for simulations with the production workloads. First, the replication overhead is more significant than that of the local cluster, because data replications tend to be more frequent on a large-scale cluster. However, it is still kept in a reasonable low level (less than 5% of the total energy consumption). Second, the battery plays a more important role in the large-scale cluster especially for the variants of GreenMR, because green energy is utilized with higher efficiency and more surplus green energy can be charged into battery. Meanwhile, as higher green supply and battery capacity are provisioned in our simulation, Figure 6(b) shows more green energy waste for the simulation study than our micro benchmark. Third, we study the detailed energy consumption along the timeline. All variants of "GreenMR" significantly reduce the peak usage of brown energy, in comparison with GreenHadoop.



(a) Brown energy consumption (b) Energy consumption breakdown

Figure 6: Overall comparison of the Facebook production workloads

Figure 7 shows the detailed energy consumption for Facebook production trace for GreenHadoop, GreenMR(BE), GreenMR(JT) and GreenMR. We can see 1) All variants of "GreenMR" significantly reduce the usage of brown energy, in comparison with GreenHadoop. 2) Even though GreenMR(BE) consumes less green energy compared to GreenHadoop, the consumption of brown energy is still reduced by considering energy efficiency. 3) GreenMR(JT) applies job transformations to utilize green energy better, therefore, the brown consumption is further reduced. 4) GreenMR reduces the brown consumption the most, because battery-assisted optimization charges the current available green energy into battery if it can be utilized more efficiently.

V. Related Work

Energy-efficient MapReduce. Recently, there have been many research studies on improving the energy efficiency of MapReduce/Hadoop. Roughly, we can categorize the related work in the following two categories.

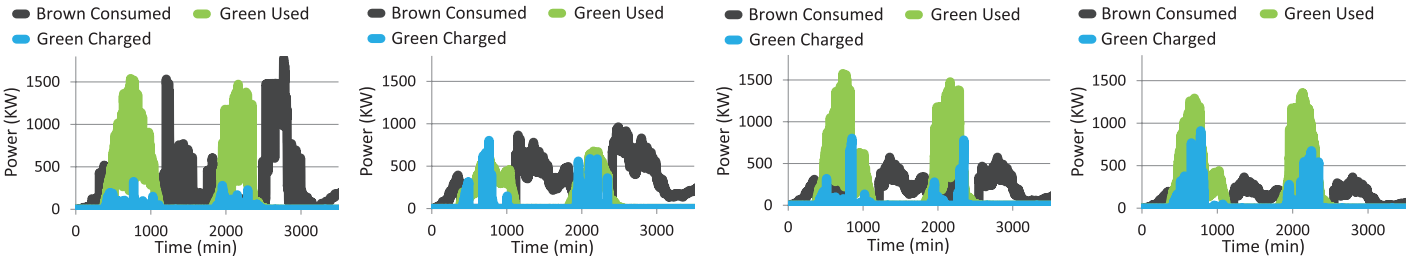
The first category is to turn down some machines in the cluster for energy saving. Different approaches have been developed to identify the machines to turn down [4], [21]. As MapReduce co-locates computation and storage in the same server, a main issue is how to place data replicas in HDFS for data availability. Many energy-aware replication strategies are proposed [22], [23]. These data placement policies are complementary to GreenMR.

The second category is to reduce the energy waste according to the non-power proportionality of data centers. Lang and Patel [17] proposed an approach called All-In Strategy (AIS), and used all the nodes in the cluster to run a batch of workloads and then powered down the entire cluster. BEEMR [24] splits the cluster into two disjoint zones, namely interactive and batch zones. The interactive zone consists of a small, fixed percentage of cluster resources and is always fully powered on. The batch zone runs workloads in a batch fashion, and is put into the low power state after the batch is finished.

Green-aware algorithms and systems. There have been many fruitful research on non-green-aware algorithms and systems for deadline-constrained job/task scheduling (e.g, [25]). Due to the dynamic feature of green energy, renewable energy aware computing has become a hot research issue in data centers. Some studies have developed models and algorithms for predictions in order to minimize the cost of a data center. Deng et al. [3] applied a two-state Lyapunov optimization to design an online control algorithm, SmartDPSS, which optimally schedules multi-source energy supply in a cost minimizing fashion. Chen et al. [26] proposed to schedule workloads across multiple geographically distributed data centers in order to utilize the renewable energy effectively. Preemption policies have been implemented on YARN [27], which further promotes the elasticity of green-aware algorithms. However, checkpointing causes further I/O contention and preemption increases complexity of schedulers, needs further study. Recently, a number of real green-aware systems are developed for different workloads. Researchers have proposed to schedule batch jobs to maximize the usage of renewable energy [28], [4], [29]. Goiri et al. has carried out a series of studies and a green data center prototype [5] to manage deferrable and non-deferrable workloads at the presence of renewable energy. Chen et al. [6] proposed ReinDB that integrates renewable energy supply into database systems on a single server. Some attention has been paid to leverage battery to store renewable energy [30], [31], [32], without considering the energy efficiency of workloads. Few of the previous studies have paid attention to the joule efficiency problem, particularly in the MapReduce/Hadoop cluster. We conjecture the concept of joule efficiency can also be applied to other applications.

VI. Conclusions

This paper proposes GreenMR by integrating energy-efficient and green-aware job/task scheduling into MapReduce. The scheduling considers the key aspects of joule efficiency in a MapReduce cluster, including energy efficiency of MapReduce workloads, renewable energy supply and battery usage. An analytical model is developed to guide scheduling decisions. We have implemented GreenMR on top of Hadoop. Our results demonstrate that GreenMR significantly reduces the brown energy usage, by 35% and 28% reduction compared with Hadoop and GreenHadoop (the state-of-the-art green-aware Hadoop) [4], respectively. Moreover, GreenMR has a



(a) GreenHadoop (b) GreenMR(BE) (c) GreenMR(JT) (d) GreenMR

Figure 7: Detailed energy consumption

lower total energy consumption, by 21% and 19% smaller than Hadoop and GreenHadoop, respectively. In the future, we will integrate the energy efficiency into our previous study which considers the performance-fairness trade-off [33].

VII. Acknowledgment

This work is supported by a MoE AcRF Tier 1 grant (MOE 2014-T1-001-145) in Singapore and a grant from the NSFC under grant NO. 61520106005 of NSFC, China. We acknowledge the support from the Singapore National Research Foundation under its Environmental & Water Technologies Strategic Research Programme and administered by the Environment & Water Industry Programme Office (EWI) of the PUB, under project 1002-IRIS-09. Zhaojie's work is in part supported by the National Research Foundation Singapore under its Interactive Digital Media (IDM) Strategic Research Programme.

References

[1] "Industry outlook: Data center energy efficiency http://www.datacenterjournal.com/industry-outlook-data-center-energy-efficiency/."

[2] R. Bianchini, "Leveraging renewable energy in data centers: present and future," in *HPDC*. ACM, 2012.

[3] W. Deng, F. Liu, H. Jin, and C. Wu, "Smartdpss: Cost-minimizing multi-source power supply for datacenters with arbitrary demand," in *ICDCS*, 2013.

[4] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: leveraging green energy in data-processing frameworks," in *Eurosys*, 2012.

[5] Í. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: Managing datacenters powered by renewable energy," in *ASPLOS*, 2013.

[6] C. Chen, B. He, X. Tang, C. Chen, and Y. Liu, "Green databases through integration of renewable energy." in *CIDR*, 2013.

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *osdi*, 2004.

[8] D. Schneider and Q. Hardy., "Under the hood at google and facebook." *Spectrum, IEEE, 48(6):63-67,*, 2011.

[9] D. S. Palasamudram, R. K. Sitaraman, B. Urgaonkar, and R. Urgaonkar, "Using batteries to reduce the power costs of internet-scale distributed networks," in *SoCC*, 2012.

[10] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri." in *OSDI*, 2010.

[11] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *MASCOTS*, 2011.

[12] S. S. Elnaffar, "Towards workload-aware dbmss: identifying workload type and predicting its change," Ph.D. dissertation, 2004.

[13] S. Jebaraj and S. Iniyan, "A review of energy models," *Renewable and Sustainable Energy Reviews*, 2006.

[14] M. Bartlett, A. M. Frisch, Y. Hamadi, I. Miguel, S. A. Tarim, and C. Unsworth, "The temporal knapsack problem and its solution," in *CPAIOR*, 2005.

[15] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *SoCC*, 2012.

[16] H. Herodotou and S. Babu, "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs," *VLDB*, 2011.

[17] W. Lang and J. M. Patel, "Energy management for mapreduce clusters," *VLDB*, 2010.

[18] G. Ananthanarayanan and R. H. Katz, "Greening the switch," in *HotPower*, 2008.

[19] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM Computer Communication Review*, 2011.

[20] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *ISCA*, 2011.

[21] R. T. Kaushik and M. Bhandarkar, "Greenhdfs: Towards an energy-conserving storage-efficient, hybrid hadoop compute cluster," in *ATC*, 2010.

[22] J. Kim and D. Rotem, "Energy proportionality for disk storage using replication," in *ICDT*, 2011.

[23] J. K. Jerry Chou and D. Rotem, "Exploiting replication for energy-aware scheduling in disk storage systems," in *ICDCS*, 2004.

[24] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale mapreduce workloads with significant interactive analysis," in *Eurosys*, 2012.

[25] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Sci. Program.*, 2006.

[26] C. Chen, B. He, and X. Tang, "Green-aware workload scheduling in geographically distributed data centers," in *CloudCom*, 2012.

[27] G. Ananthanarayanan, C. Douglas, R. Ramakrishnan, S. Rao, and I. Stoica, "True elasticity in multi-tenant data-intensive compute clusters," in *SoCC*, 2012.

[28] Í. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenslot: scheduling energy consumption in green datacenters," in *SC*, 2011.

[29] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers," *SIGOPS*, 2012.

[30] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations of tapping into stored energy for datacenters," in *ISCA*, 2011.

[31] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Aggressive datacenter power provisioning with batteries," *ACM Transactions on Computer Systems (TOCS)*, 2013.

[32] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy, "Energy storage in datacenters: what, where, and how much?" in *ACM SIGMETRICS Performance Evaluation Review*, 2012.

[33] Z. Niu, S. Tang, and B. He, "Gemini: An adaptive performance-fairness scheduler for data-intensive cluster computing," *CloudCom*, 2015.