GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks

Íñigo Goiri
Dept. of Computer Science
Rutgers University
goiri@cs.rutgers.edu

Kien Le
Dept. of Computer Science
Rutgers University
lekien@cs.rutgers.edu

Thu D. Nguyen
Dept. of Computer Science
Rutgers University
tdnguyen@cs.rutgers.edu

Jordi Guitart
Universitat Politècnica de Catalunya
Barcelona Supercomputing Center
jguitart@ac.upc.edu

Jordi Torres
Universitat Politècnica de Catalunya
Barcelona Supercomputing Center
torres@ac.upc.edu

Ricardo Bianchini
Dept. of Computer Science
Rutgers University
ricardob@cs.rutgers.edu

Abstract

Interest has been growing in powering datacenters (at least partially) with renewable or "green" sources of energy, such as solar or wind. However, it is challenging to use these sources because, unlike the "brown" (carbon-intensive) energy drawn from the electrical grid, they are not always available. This means that energy demand and supply must be matched, if we are to take full advantage of the green energy to minimize brown energy consumption. In this paper, we investigate how to manage a datacenter's computational workload to match the green energy supply. In particular, we consider data-processing frameworks, in which many background computations can be delayed by a bounded amount of time. We propose GreenHadoop, a MapReduce framework for a datacenter powered by a photovoltaic solar array and the electrical grid (as a backup). GreenHadoop predicts the amount of solar energy that will be available in the near future, and schedules the MapReduce jobs to maximize the green energy consumption within the jobs' time bounds. If brown energy must be used to avoid time bound violations, GreenHadoop selects times when brown energy is cheap, while also managing the cost of peak brown power consumption. Our experimental results demonstrate that GreenHadoop can significantly increase green energy consumption and decrease electricity cost, compared to Hadoop.

EuroSys'12, April 10–13, 2012, Bern, Switzerland.

Categories and Subject Descriptors D.4.1 [*Operating Systems*]: Process Management—Scheduling

Keywords Green energy; renewable energy; MapReduce; energy-aware scheduling; cost-aware scheduling

1. Introduction

It is well-known that datacenters consume an enormous amount of power [31], representing a financial burden for their operating organizations, an infrastructure burden on power utilities, and an environmental burden on society. Large Internet companies (e.g., Google and Microsoft) have significantly improved the energy efficiency of their multi-megawatt datacenters. However, the majority of the energy consumed by datacenters is actually due to countless small and medium-sized ones [31], which are much less efficient. These facilities range from a few dozen servers housed in a machine room to several hundreds of servers housed in a larger enterprise installation.

These cost, infrastructure, and environmental concerns have prompted some datacenter operators to either generate their own solar/wind energy or draw power directly from a nearby solar/wind farm. Many small and medium datacenters (partially or completely) powered by solar and/or wind energy are being built all over the world (see *http://www.eco-businesslinks.com/green_web_hosting.htm* for a partial list). This trend will likely continue, as these technologies' capital costs continue to decrease (e.g., the cost of solar energy has decreased by 7-fold in the last two decades [29]) and governments continue to provide incentives for green power generation and use (e.g., federal and state incentives in New Jersey can reduce capital costs by 60% [7]).

For the scenarios in which green datacenters are appropriate, we argue that they should connect to both the solar/wind energy source and the electrical grid, which acts as a backup

when green energy is unavailable. The major challenge with solar or wind energy is that, unlike brown energy drawn from the grid, it is not always available. For example, photovoltaic (PV) solar energy is only available during the day and the amount produced depends on the weather and the season.

To mitigate this variability, datacenters could "bank" green energy in batteries or on the grid itself (called net metering). However, these approaches have many problems: (1) batteries incur energy losses due to internal resistance and self-discharge; (2) battery-related costs can dominate in solar-powered systems [11]; (3) batteries use chemicals that are harmful to the environment; (4) net metering incurs losses due to the voltage transformation involved in feeding the green energy into the grid; (5) net metering is unavailable in many parts of the world; and (6) where net metering is available, the power company may pay less than the retail electricity price for the green energy. Given these problems, the best way to take full advantage of the available green energy is to match the energy demand to the energy supply.

Thus, in this paper, we investigate how to manage the computational workload to match the green energy supply in small/medium datacenters running data-processing frameworks. In particular, we consider the MapReduce framework [6] and its Hadoop implementation [4]. Data-processing frameworks are an interesting target for our research, as they are popular and often run many low-priority batch processing jobs, such as background log analysis, that do not have strict completion time requirements; they can be delayed by a bounded amount of time. However, scheduling the energy consumption of MapReduce jobs is challenging, because they do not specify the number of servers to use, their run times, or their energy needs. Moreover, power-managing servers in these frameworks requires guaranteeing that the data to be accessed by the jobs remains available.

With these observations in mind, we propose GreenHadoop, a MapReduce framework for datacenters powered by PV solar arrays and the electrical grid (as a backup). GreenHadoop seeks to maximize the green energy consumption of the MapReduce workload, or equivalently to minimize its brown energy consumption. GreenHadoop predicts the amount of solar energy that will likely be available in the future, using historical data and weather forecasts. It also estimates the approximate energy needs of jobs using historical data. Using these predictions, GreenHadoop may then decide to delay some (low-priority) jobs to wait for available green energy, but always within their time bounds. If brown energy must be used to avoid bound violations, it schedules the jobs at times when brown energy is cheap, while also managing the cost of peak brown power consumption. GreenHadoop controls energy usage by using its predictions and knowledge of the data required by the scheduled jobs. With this information, it defines how many and which servers to use; it transitions other servers to low-power states to the extent possible.

We evaluate GreenHadoop using two realistic workloads running on a 16-server cluster. We model the datacenter's solar array as a scaled-down version of an existing Rutgers solar farm. The brown energy prices and peak brown power charges are from a power company in New Jersey. We compare GreenHadoop's green energy consumption and brown electricity cost to those of standard Hadoop and of an energy-aware version of Hadoop that we developed. Our results demonstrate that GreenHadoop can increase green energy consumption by up to 31% and decrease brown electricity cost by up to 39%, compared to Hadoop. In addition, our results show that GreenHadoop is robust to workload variability and effective for a range of time bounds.

GreenHadoop is most closely related to our own GreenSlot [9], a green energy-aware scheduler for scientific computing jobs. However, GreenSlot relies on extensive user-provided information about job behavior, assumes that persistent data is always available to jobs regardless of the servers' power states, and does not manage the cost of peak brown power consumption. In contrast, GreenHadoop requires no job behavior information, and explicitly manages data availability and peak brown power costs.

We conclude that green datacenters and software that is aware of the characteristics of both green and brown electricities can have a key role in building a more sustainable and cost-effective Information Technology (IT) ecosystem.

In summary, we make the following contributions:

- We introduce GreenHadoop, a MapReduce framework for datacenters partly powered by solar energy;
- We introduce MapReduce job scheduling and data management techniques that are aware of green energy, brown energy prices, and peak brown power charges;
- We demonstrate that it is possible to manage green energy use and brown electricity cost when the jobs' run times and energy demands are not specified; and
- We present extensive results isolating the impact of different aspects of the implementation.

2. Background

Use of solar energy in datacenters. Solar and wind are two of the most promising green energy technologies, as they do not cause the environmental disruption of hydroelectric energy and do not have the waste storage problem of nuclear energy. Solar/wind equipment produces Direct Current (DC) electricity, which is typically converted to Alternating Current (AC) by DC/AC inverters.

In this paper, we assume that the datacenter generates its own PV solar energy. (Except for our solar energy predictions, our work is directly applicable to wind energy as well.) Self-generation is attractive for multiple reasons, including (1) the fact that energy losses with power transformation and transmission can exceed 40%; (2) the ability to survive grid outages, which are common in some developing countries;

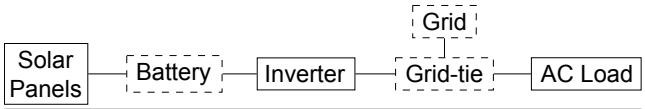

Figure 1. Components of a (partially) solar-powered computer system. Dashed boxes represent optional components.

(3) the fact that PV power scales poorly, as the cost/W does not decrease beyond 800KW (the maximum inverter size today [28]); and (4) the ability to eventually lower costs. In fact, based on the results from Section 5 and the governmental incentives in New Jersey, the current capital cost of installing solar panels for the datacenter we model can be amortized by savings in brown energy cost in 10.6 years of operation. This amortization period is substantially shorter than the typical 20-30 years lifetime of the panels. The period will be even shorter in the future, as solar costs continue to decrease at a rapid pace [29]. The increasing popularity of distributed generation and microgrids suggests that many people find self-generation attractive.

There are multiple ways to connect solar panels to a datacenter. Figure 1 shows a general setup. The solar panels can be connected to batteries for storing excess energy during periods of sunlight and discharging it during other periods. The datacenter must also be connected to the electrical grid via a grid-tie device if it must be operational even when solar energy is not available. Where net metering is available, it is possible to feed excess solar energy into the grid for a reduction in brown energy costs.

The design we study does not include batteries or net metering for the reasons mentioned in the Introduction. Thus, any green energy not immediately used is wasted. Fortunately, GreenHadoop is very successful at limiting waste.

Brown energy prices and peak brown power charges. Datacenters often contract with their power companies to pay variable brown energy prices, i.e. different dollar amounts per kWh of consumed brown energy. The most common arrangement is for the datacenter to pay less for brown energy consumed during an off-peak period (e.g., at night) than during an on-peak period (e.g., daytime). Thus, it would be profitable for the datacenter to schedule part of its workload during the off-peak periods if possible.

However, high brown energy costs are not the only concern. Often, datacenters also have to pay for their peak brown power consumption, i.e. a dollar amount per kW of brown power at the highest period of brown power usage. Even though these charges have almost always been overlooked in the literature, the peak brown power charge can be significant, especially during the summer. Govindan *et al.* estimate that this component can represent up to 40% of the overall electricity cost of a datacenter [10].

To compute the peak charges, utilities typically monitor the average brown power consumption within 15-minute windows during each month. They define the maximum of these averages as the peak brown power for the month.

MapReduce and Hadoop. MapReduce is a framework for processing large data sets on server clusters [6]. Each MapReduce program defines two functions: map and reduce. The framework divides the input data into a set of blocks, and runs a *map task* for each block that invokes the map function on each key/value pair in the block. The framework groups together all intermediate values produced by the map tasks with the same intermediate key. It then runs the *reduce tasks*, each of which invokes the reduce function on each intermediate key and its associated values from a distinct subset of generated intermediate keys. The reduce tasks generate the final result.

Hadoop is the best-known, publicly available implementation of MapReduce [4]. Hadoop comprises two main parts: the Hadoop Distributed File System (HDFS) and the Hadoop MapReduce framework. Data to be processed by a MapReduce program is stored in HDFS. HDFS splits files across the servers' local disks. A cluster-wide NameNode process maintains information about where to find each data block.

Users submit *jobs* to the framework using a client interface. This interface uses each job's configuration parameters to split the input data and set the number of tasks. Jobs must identify all input data at submission time. The interface submits each job to the JobTracker, a cluster-wide process that manages job execution. Each server runs a configurable number of map and reduce tasks concurrently in compute "slots". The JobTracker communicates with the NameNode to determine the location of each job's data. It then selects servers to execute the jobs, preferably ones that store the needed data if they have slots available. Hadoop's default scheduling policy is FIFO.

3. MapReduce in Green Datacenters

We propose GreenHadoop, a data-processing framework for datacenters powered by PV solar panels and the electricity grid. GreenHadoop relies on predictions of the availability of solar energy, and a scheduling and data availability algorithm that is aware of green energy, brown energy prices, and peak brown power charges. We refer to the overall brown energy and power costs as the brown electricity cost.

To achieve its goals of maximizing green energy usage and minimizing brown electricity cost, GreenHadoop may delay the execution of some jobs. To avoid excessive delays, GreenHadoop attempts to complete all jobs within a bounded amount of time from their submissions. GreenHadoop is beneficial because datacenters are often underutilized and many jobs have loose performance requirements (e.g., data and log analysis, long simulations, jobs submitted on Friday whose output is not needed until Monday).

Figure 2 illustrates the behavior of GreenHadoop (bottom), in comparison to conventional Hadoop (top) and an energy-aware version of Hadoop (middle) for three MapReduce jobs. Hadoop executes the jobs immediately when they arrive, using all servers to complete the jobs as quickly as

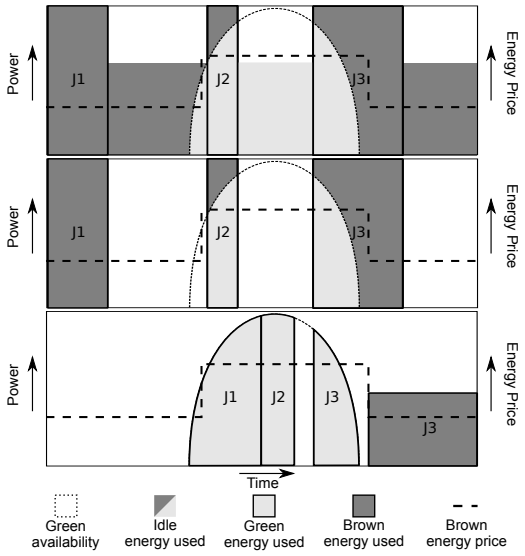

Figure 2. Scheduling 3 MapReduce jobs (J1-J3) with Hadoop (top), energy-aware Hadoop (middle), and GreenHadoop (bottom).

possible. Hadoop keeps all servers active even if they are idle. As a result, Hadoop wastes substantial green and brown energy, and incurs unnecessary energy costs. In contrast, the energy-aware Hadoop that we implemented reduces waste by transitioning idle servers to a low-power state.

GreenHadoop behaves differently. It uses as many servers as green energy can sustain when it is available, fewer servers (if possible) when brown energy is cheap, and even fewer (if at all necessary) when brown energy is expensive. Figure 2(bottom) shows that GreenHadoop delayed jobs J1 and J2 to maximize the green energy consumption. More interestingly, GreenHadoop executed part of J3 with green energy, and delayed the other part until the brown energy became cheaper. Moreover, GreenHadoop did not use all servers to run J3 with brown energy to limit the peak brown power costs. When certain servers need not be fully active, GreenHadoop transitions them to lower power states.

Essentially, GreenHadoop determines how many and which servers to use at each point in time, and schedules the jobs on those servers. The other servers can be deactivated to conserve energy.

We designed GreenHadoop as a wrapper around a modified version of Hadoop. The wrapper implements the scheduling, data management, and prediction of solar energy availability described in the remainder of the section. We also briefly discuss our modest changes to Hadoop.

3.1 Scheduling and Data Availability

3.1.1 Overview

At submission time, users can specify the priority for their jobs. Like standard Hadoop, GreenHadoop has five priority classes: very high, high, normal, low, and very low. GreenHadoop executes very high and high priority jobs as soon as possible, giving them all the servers they can use. In contrast, GreenHadoop may delay some of the normal, low, and very low priority jobs by a bounded amount of time (by default, at most one day in our experiments). These behaviors reflect our performance goals: high-priority jobs should complete as quickly as in standard Hadoop, whereas low-priority ones must complete within their time bounds.

GreenHadoop divides time into fixed-size "epochs" (four minutes in our experiments). At the beginning of each epoch, GreenHadoop determines whether the number of active servers should be changed and whether all the data needed by the scheduled jobs is available. Specifically, servers can be in one of three states in GreenHadoop: Active, Decommissioned, or Down (ACPI's S3 state). In the Decommissioned state, no new tasks are started on the server, but previously running tasks run to completion. Moreover, no new blocks are stored at a decommissioned server, but it still serves accesses to the blocks it currently stores. Whenever servers are not needed for computation, GreenHadoop first transitions them to the Decommissioned state, and then later sends them to the Down state. To prevent data unavailability, it replicates any data needed by scheduled jobs from the decommissioned servers before sending them down. Every so often, GreenHadoop reduces the amount of replication.

Estimating the jobs' energy and time requirements is challenging. Unlike traditional batch job schedulers, e.g. [8, 32], GreenHadoop does not have information about the jobs' desired number of servers or expected running times, from which we could estimate their energy needs. Instead, GreenHadoop computes average running times and energy consumptions based on prior history, relying on aggregate statistics of the datacenter's workload, rather than information about specific jobs or applications. These averages are then used to estimate the resource requirements of groups of jobs, not each individual job, during the "scheduling horizon" (one day ahead in our experiments).

GreenHadoop is electricity-cost-aware in that it favors scheduling jobs in epochs when energy is cheap. To prioritize green energy, it is assumed to have zero cost. When the price of brown energy is not fixed and brown energy must be used, GreenHadoop favors the cheaper epochs. In addition, GreenHadoop manages the peak brown power usage by limiting the number of active servers, if that can be done without violating any time bounds.

3.1.2 Algorithm Details

Figure 3 shows the pseudo-code of the GreenHadoop algorithm. As line 1 suggests, GreenHadoop maintains two job queues: Run and Wait. The Run queue is implemented by Hadoop, whereas we implement the Wait queue entirely in the wrapper. Jobs submitted with very high or high priority are sent straight to the Hadoop queue in FIFO order, whereas others initially go to the Wait queue also in FIFO order.

Assigning deadlines. Lines 2-26 describe the system's behavior at the beginning of each epoch. First (line 3), it as-

```
0.  At job submission time:
1.     Very high and high priority jobs go straight to the Run queue; other jobs go to the Wait queue

2.  At the beginning of each epoch:
3.     Assign internal latest-start deadlines to all waiting jobs that arrived in the previous epoch
4.     Calculate the number of active servers to use during the scheduling horizon (defined in lines 11-18)
5.     Select waiting jobs to move to the Run queue (in line 10):
6.        If a job is about to violate its latest-start deadline, select the job
7.        If a job has all the data it needs available, select the job
8.        If the active servers are not fully utilized, select the jobs that require data access to the fewest servers
9.     Manage the servers' states and data availability (defined in lines 19-26)
10.    When the data required by the selected jobs is available, move them to the Run queue

11. Calculate the number of active servers to use:
12.    Calculate the dynamic energy required by all running and waiting jobs
13.    Compute/predict the energy available for each type of energy during the scheduling horizon
14.    Subtract the static energy of the entire system from the energy available
15.    Subtract the dynamic energy required by any very high or high priority jobs from the energy available
16.    Assign the remaining green energy to waiting jobs
17.    Assign the remaining brown energy and power to waiting jobs, considering brown energy prices and peak brown power charges
18.    If some waiting jobs have not been assigned energy, reject new jobs that arrive in the next epoch

19. Manage the servers' states and data availability:
20.    Ensure that all the data required by the selected jobs is available in active or decommissioned servers
21.    If some data is not available, turn into Decommissioned state the min set of down servers containing the data
22.    If current number of active servers is smaller than desired number, transition servers from Decommissioned to Active state
23.       If we still need more active servers, transition them from Down state to Active state
24.    If current number of active servers is larger than desired number, transition active servers to Decommissioned state
25.    Replicate the data from servers in Decommissioned state, if necessary
26.    Check if servers that are in Decommissioned state can be sent to Down state
```

Figure 3. GreenHadoop algorithm for scheduling and data availability.

signs internal "latest-start deadlines" to any waiting jobs that were queued in the previous epoch. The latest-start deadline specifies the latest epoch in which the job needs to start, so that it is expected to complete within the system's desired completion time bound. GreenHadoop computes the latest-start deadline for each job based on its arrival epoch, the expected duration of the group of jobs after (and including) it in the Wait queue, and the completion time bound.

Calculating the number of active servers. Second (lines 4, 11-18), GreenHadoop calculates the number of active servers to use in each epoch during the scheduling horizon. This calculation involves computing the dynamic energy required by the jobs currently in the system. GreenHadoop computes this energy by multiplying the average dynamic energy per job by the number of waiting jobs and adding the leftover dynamic energy required by the running jobs. GreenHadoop estimates the average dynamic energy per job using an exponentially weighted moving average of the dynamic energy consumed during the last ten epochs divided by the number of jobs executed in this period.

After computing the dynamic energy required by the jobs, GreenHadoop assigns the available green, cheap brown, and expensive brown energies to them (lines 13-17). GreenHadoop predicts the green energy that will be available (see Section 3.2). It computes the brown energy available assuming that all servers could be active during the (pre-defined) periods of cheap and expensive brown energy. From these three types of energy, GreenHadoop subtracts the static energy (corresponding to the energy consumed by all servers in Down state, the switch, and an always-on server) to be consumed. In addition, it subtracts the dynamic energy needed by any very high or high priority jobs in the Run queue.

In assigning the remaining energy to jobs, GreenHadoop first assigns the green energy, then the cheap brown energy, and finally the expensive brown energy. In the absence of peak brown power charges, it uses all the servers to consume the brown energies. Otherwise, it records the peak brown power that has been reached so far this month. We refer to this quantity as "past-peak-brown-power". It then finds the best brown power at which to limit consumption between past-peak-brown-power and the maximum achievable power (i.e., all machines activated). Increasing the peak brown power increases peak charges, but those charges may be offset by reduced runtime in periods of expensive brown energy. The best peak brown power is the one that provides enough energy to complete the schedule and leads to the minimum electricity cost.

The number of active servers during each epoch within the horizon derives from the amount of dynamic energy assigned to the epoch. Since we know the length of the epoch, the dynamic energy can be transformed into an average dynamic power, which in turn can be transformed into a number of active servers. For example, suppose that the average available dynamic power in an epoch is 1000W. Suppose also that we currently have 3 active servers running jobs and serving data, and 3 decommissioned servers just serving data; the other servers are down and, thus, consume no dynamic power. In our experiments, active servers and decommissioned servers consume at most roughly 136W and 66W of dynamic power, respectively. Thus, the current total dynamic power consumption is $606W = 3 \times 136W + 3 \times 66W$, leaving 394W available. In response, GreenHadoop would activate the 3 decommissioned servers and 1 down server for a total of $952W = 606W + 3 \times (136W - 66W) + 136W$;

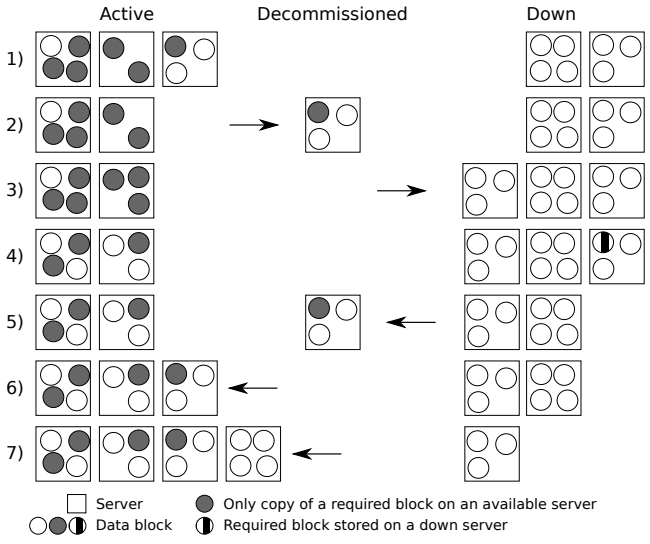

Figure 4. Server power state and data management sequence. 1) Initial configuration; 2) GreenHadoop does not need one of the active servers and sends the active server with the fewest data blocks required by the jobs in the Run queue to the Decommissioned state; 3) GreenHadoop replicates the required block from the decommissioned server to an active server, and then transitions the decommissioned server to the Down state; 4) A new job is moved to the Run queue, requiring data stored only on a down server; 5) GreenHadoop transitions the down server to Decommissioned to provide the needed data; 6) GreenHadoop needs one more active server, transitioning the decommissioned server to the Active state; 7) GreenHadoop still needs another server, and so it activates the down server with the most data.

the remaining 48W would not be usable. If some waiting jobs cannot be assigned energy within the scheduling horizon (line 18), the system is overcommitted, so GreenHadoop rejects the jobs arriving in the next epoch.

Selecting waiting jobs to move to the Run queue. Third (lines 5-8), GreenHadoop selects some waiting jobs that will soon be moved to the Run queue. GreenHadoop considers each waiting job in order and selects the job if (1) it is about to violate its latest-start deadline, (2) all the data it needs is on active or decommissioned servers, or (3) an active server is under-utilized (i.e., not all of its compute slots are being used) and the job requires the fewest down servers to be brought up for its data. GreenHadoop moves the selected jobs to the Run queue when the data they require becomes available (line 10). Jobs selected due to their latest-start deadlines are sent to the Run queue with very high priority.

Managing server power states and data availability. Finally, GreenHadoop manages the servers' states and the data availability (lines 9, 19-26). If not all data required by the selected jobs is currently available in active or decommissioned servers, GreenHadoop transitions the needed servers from the Down state to the Decommissioned state (lines 20 and 21). It then adjusts the number of active servers based on the target number computed earlier (lines 22-26).

Figure 4 illustrates an example of GreenHadoop managing its servers' power states and data availability. When it reduces the number of active servers (step 2), GreenHadoop splits them into two groups: (a) those that have executed tasks of still-running jobs, and (b) those that have not. Then, it sorts each group in ascending order of the amount of data stored at the server that is required by jobs in the Run queue. Finally, it starts sending servers in group (b) to the Decommissioned state, and then moves to group (a), stopping when the target reduction has been reached.

After adjusting the number of active servers, GreenHadoop considers sending decommissioned servers to the Down state (step 3). A decommissioned server cannot go down if it has executed a task of still-running jobs. Moreover, before the server can go down, GreenHadoop must copy (replicate) any data that is required by a job in the Run queue but that is not currently available on the active servers.

In the opposite direction, when servers need to be transitioned from Decommissioned to Active (step 6), GreenHadoop divides the decommissioned servers into the same two groups as above. Then, it sorts each group in decreasing order of the amount of data stored at the server that is required by the jobs in the Run queue. It starts activating the decommissioned servers in group (a) and then moves to group (b), stopping if the target number is reached. If more active servers are needed (i.e., the decommissioned servers are not enough to reach the target), GreenHadoop activates the servers in Down state that store the most data required by jobs in the run queue (step 7).

3.1.3 Power State and Data Management Rationale

GreenHadoop relies on dynamic replication of data blocks to guarantee data availability when servers are turned off. However, it is possible to manage power states and data availability without dynamic replication, as in the covering subset approach [20]. This approach ensures that at least one copy of every data block stored in a Hadoop cluster is kept on a designated subset of servers (the covering subset); keeping these servers in either the Active or Decommissioned state would statically guarantee the availability of all data.

Unfortunately, using the covering subset approach would limit GreenHadoop's ability to send servers to the Down state. For example, if each data block is replicated 3 times—a typical replication level—then to keep storage balanced across the cluster, roughly 1/3 of the servers would have to be in the covering subset. The servers in the covering subset cannot be sent to the Down state, even if they are not needed for running computations.

In contrast, GreenHadoop guarantees availability only for data required by jobs in the Run queue. When the load is low, e.g. only one job is running, the required data is typically a small subset of all data stored in the cluster, enabling GreenHadoop to send all but one server to the Down state. Thus, our approach trades off the energy required to replicate (and later dereplicate) data against the energy savings from

transitioning servers to the Down state when they are not needed for computation. This tradeoff is advantageous since (1) current servers are not power-proportional, so a server that is mostly idle only servicing data requests requires a large fraction of its peak power; (2) transitions to and from the S3 state take on the order of only several seconds; (3) GreenHadoop transitions servers to S3 in an order that tries to maximize energy savings; (4) replicating a data block is typically a cheap operation compared to the duration and power consumption of the compute task that operates on it; and (5) dereplication is nearly free of cost. We evaluate GreenHadoop's data management approach in Section 5.

3.2 Predicting the Availability of Solar Energy

GreenHadoop can easily use any model that predicts the availability of green energy. In fact, it would even adapt to wind energy predictions without modification. Our current implementation uses the model introduced by Sharma *et al.* [26] to predict solar energy. This model is based on the simple premise that energy generation is inversely related to the amount of cloud coverage, and is expressed as: $E_p(t) = B(t)(1 - \mathit{CloudCover})$, where $E_p(t)$ is the amount of solar energy predicted for time t, $B(t)$ is the amount of solar energy expected under ideal sunny conditions, and *CloudCover* is the forecasted percentage cloud cover (given as a fraction between 0 and 1).[1]

We implement solar energy prediction using the above model at the granularity of an hour. We use weather forecasts from Intellicast.com, which provides hourly predictions that include cloud coverage for up to 48 hours into the future. We use historical data to instantiate $B(t)$. We compute a distinct $B(t)$ for each month of the year to account for seasonal effects. For each month, we set $B(t)$ to the actual energy generated by the day with the highest energy generation from the same month of the previous year. (For new installations, it is also possible to use data from the previous month.)

Unfortunately, weather forecasts can be wrong. For example, we have observed that predictions of thunderstorms are frequently inaccurate and can remain inaccurate throughout a day. Furthermore, weather is not the only factor that affects energy generation. For example, after a snow storm, little energy will be generated while the solar panels remain covered by snow even if the weather is sunny.

To increase accuracy in these hard-to-predict scenarios, we use an alternate method of instantiating *CloudCover* proposed by Goiri *et al.* [9]. Specifically, we assume that the recent past can predict the near future [26], and compute *CloudCover* using the observed energy generated in the previous hour. When invoked, our prediction module compares the accuracy of the two prediction methods for the last hour and chooses the more accurate method to instantiate *CloudCover* for the remainder of the current day. Beyond the current day, we instantiate *CloudCover* using weather forecasts.

[1] Factors other than cloud cover, such as temperature, can affect the amount of solar energy produced. However, Section 5 shows that ignoring these factors still leads to sufficient accuracy for our purposes.

3.3 Modified Hadoop

As already mentioned, most of GreenHadoop is implemented in a wrapper external to Hadoop. However, we also extended Hadoop itself with power management functionality. Our main changes include (1) the introduction of the Decommissioned and Down states for servers, which involved changes to the JobTracker and NameNode processes; (2) enabling the NameNode process to know what data is present in down servers, so that they can be activated when their data is not replicated elsewhere; and (3) improvements to the block replication and replica removal functionality already present in Hadoop.

4. Evaluation Methodology

Hardware and software. We evaluate GreenHadoop using a 16-server cluster, where each server is a 4-core Xeon machine with 8GB RAM and a 7200-rpm SATA disk with 64GB of free space for data. The servers are inter-connected by a Gigabit Ethernet switch.

GreenHadoop extends Hadoop version 0.21 for Linux with roughly 5000 uncommented lines of Python wrapper code, and adds around 100 lines to Hadoop itself. We study 3 versions of GreenHadoop: "GreenOnly", which makes decisions based on knowledge about green energy availability; "GreenVarPrices", which considers both green energy and variable brown energy prices, and "GreenVarPricesPeak" which considers green energy, variable brown energy prices, and peak brown power charges. The GreenVarPricesPeak version is the full-blown GreenHadoop.

For comparison, we also study "Hadoop", the regular Hadoop FIFO scheduler; and "Energy-Aware Hadoop" ("EAHadoop"), our own extension of Hadoop that manages data availability and transitions servers to lower power states when they are not required. EAHadoop is simply a version of GreenHadoop that disregards green energy, brown energy prices, and peak brown power charges. In fact, EAHadoop's pseudo-code is that in Figure 3 minus lines 13, and 15-17.

Workloads. We study two widely differing workloads: (1) a synthetic workload, called "Facebook-Derived" or simply "FaceD", that models Facebook's multi-user production workload [33]; and (2) the Web indexing part of the Nutch Web-search system [5], called "Nutch Indexing" or simply "NutchI". FaceD contains jobs with widely varying execution time and data set sizes, representing a scenario where the cluster is used to run many different types of applications. NutchI jobs are relatively homogeneous, representing a scenario where the cluster is used to run a continuous production batch workload. By default, we submit all jobs of these workloads with normal priority to provide the maximum flexibility to GreenHadoop. We study the impact of different mixes of priorities in Section 5.

Cat.	% Jobs	# Maps	# Reds	In (GB)	Out (GB)
0	59.0%	4	1	0.25	0.01
1	9.8%	10	1-2	0.63	0.03
2	8.7%	20	1-5	1.25	0.06
3	8.5%	40	2-10	2.50	0.13
4	5.7%	80	4-20	5.00	0.25
5	4.4%	150	8-38	9.38	0.48
6	2.5%	300	15-75	18.75	0.95
7	1.3%	600	30-150	37.50	1.90

Table 1. FaceD workload characteristics.

FaceD is a scaled-down version of the workload studied in [33] because our cluster is significantly smaller. As in that paper, we do not run the Facebook code itself; rather, we mimic the characteristics of its jobs using "loadgen." Loadgen is a configurable MapReduce job from the Gridmix benchmark included in the Hadoop distribution. We scale the workload down in two ways: we reduce the number of maps by a factor of 4, and eliminate the largest 1.7% of jobs.

Table 1 summarizes FaceD, which comprises 8 categories of jobs (leftmost column). The 2nd column lists the fraction of the jobs that corresponds to each category. The 3rd and 4th columns list the number of maps and reduces in each job. The length of each map task is uniformly distributed between 9s and 60s. Each reduce task takes between 2s and 12s per map in the job. The 5th and 6th columns list the amount of input and output data that each job handles. Each map task operates on a data block of 64MB (the default Hadoop block size), and each reduce task outputs roughly 13MB. On average, each job touches 2.5GB of data with a minimum of 250MB and a maximum of 37.5GB. Jobs arrive at the cluster continuously according to a Poisson distribution with mean inter-arrival time of 30s. These parameters lead to a cluster utilization of 56% in Hadoop, which is substantially higher than those typically seen in the small and medium-sized datacenters that we target [25]. We investigate the impact of utilization in Section 5.

NutchI consists of jobs that index groups of pages previously fetched from our Web domain. Each job runs 42 map tasks and 1 reduce task. Each map task takes either 4s or 12s, whereas the reduce tasks take 50s. On average, each job touches 85MB of data. Jobs arrive according to a Poisson distribution with mean inter-arrival time of 20s. These characteristics lead to a cluster utilization of 35%.

Power consumption and solar panel array. We measured the power consumption of a server running Hadoop jobs using a Yokogawa multimeter. A Down server consumes 9W of static power, whereas a Decommissioned server consumes 75W (9W static + 66W dynamic). A server executing 1 to 4 tasks consumes 100W, 115W, 130W, and 145W, respectively. Together with the 55W consumed by the switch, the common-case peak power consumption of our system for our workloads is $2375W = 16 \times 145W + 55W$. Transitioning into and out of Down state takes 9 seconds.

We model the solar panel array as a scaled-down version of the Rutgers solar farm, which can produce 1.4MW of power. Specifically, we estimate the production of the smaller installation by scaling the farm's actual energy production over time compared its maximum capacity. We scale the farm's production down to 14 solar panels capable of producing 3220W. We select this scaled size because, after derating, it produces roughly the common-case peak power consumption of our system.

We considered more than one year worth of solar energy production by the farm starting on March 8, 2010. From this set, we picked 5 pairs of consecutive days to study in detail. Each pair represents a different pattern of solar energy production for the 1st and 2nd days, and was chosen randomly from the set of consecutive days with similar patterns. For example, days 05/09/11 and 05/10/11 represent two consecutive days with high solar energy production. We refer to this pair as "High-High" and use it as our default. For the "High-Low", "Low-High", "Low-Low", and "Very Low-Very Low" scenarios, we use the pairs of days starting on 05/12, 06/14, 06/16, and 05/15 in 2011, with 43.2, 29.8, 32.4, 25.9, and 8.4 kWh of generated solar energy, respectively. We study these pairs of days because they correspond to different weather patterns that affect both the amount of energy produced and the accuracy of our energy production predictor. In particular, predictions are typically very accurate for sunny days with "High" energy production, somewhat less accurate for cloudy days with "Very Low" energy production, and possibly much less accurate for partially cloudy days with "Low" energy production (see Section 5).

Brown electricity pricing. We assume on-peak/off-peak pricing, the most common type of variable brown energy pricing. In this scheme, energy costs less during off-peak times (11pm to 9am) and more during on-peak times (9am to 11pm). The difference in prices is largest in the summer (June-September). We assume the energy prices charged by PSEG in New Jersey: \$0.13/kWh and \$0.08/kWh (summer) and \$0.12/kWh and \$0.08/kWh (rest of year). We also assume the peak brown power charges from PSEG: \$13.61/kW (summer) and \$5.59/kW (rest of year). Winter pricing applies to the High-High and High-Low days.

Accelerating the experiments. To fully observe the behavior of GreenHadoop and its versions in real time, we would have to execute each of our experiments for at least one entire day. This would allow us to exactly reproduce the periods of solar energy production ($\sim$10 hours), on-peak/off-peak energy prices (14 and 10 hours, respectively), and peak power accounting (15 minutes). However, it would also entail 62 days of non-stop experiments, which would be infeasible. Thus, to speed up our study while demonstrating GreenHadoop's full functionality, we run the unchanged FaceD and NutchI workloads, while shortening the above three periods, GreenHadoop's epoch, and GreenHadoop's

	Prediction Error (%)					
	1	**3**	**6**	**12**	**24**	**48**
Average	12.6	17.6	21.6	21.4	21.2	20.6
Median	11.0	16.4	20.4	20.4	20.3	17.5
90th percentile	22.2	28.9	34.0	37.4	37.9	37.8

Table 2. Error when predicting 1, 3, 6, 12, 24, and 48 hours ahead.

horizon by a factor of 24. This shortening factor means that each of our 2-day experiments executes in just 2 hours.

GreenHadoop's scheduling and data availability algorithm itself cannot be accelerated. For FaceD, it takes a maximum of 0.4 seconds (without any optimizations) to prepare a schedule on one of our servers. This maximum occurs when the largest number of jobs (40) is in the Wait queue. Data replication and de-replication also cannot be accelerated. This presents the worse-case scenario for GreenHadoop because the energy consumption of these activities are amortized across a much shorter period.

5. Evaluation Results

This section presents our experimental results. First, we evaluate the accuracy of our solar energy predictions. Second, we compare GreenOnly, GreenVarPrices, and GreenVarPricesPeak in turn with Hadoop and EAHadoop to isolate the benefits of being aware of green energy, being aware of brown energy prices, and being aware of peak brown power charges. Third, we consider the impact of various parameters, including the amount of green energy available, datacenter utilization, fraction of high priority jobs, and shorter time bounds. Fourth, we study the accuracy of GreenVarPricesPeak's energy estimates. Finally, we compare the GreenVarPricesPeak results for FaceD and NutchI.

Throughout these experiments, none of the systems we study violates any job time bounds except when we explore time bounds of 12 hours or less.

Predicting solar energy. Table 2 shows the percentage prediction error for daily energy production when predicting 1 to 48 hours ahead for the two months that include the 10 days used in our evaluation. We compute this error as the sum of the absolute difference between the predicted value and actual energy production for each hour in a day, divided by the ideal daily production (i.e., $\sum_{t=0}^{23} B(t)$).

These results show that the predictions are reasonably accurate, achieving median and 90^{th} percentile errors of 11.0% and 22.2%, respectively, when predicting energy production for the next hour. The predictions tend to be less accurate for Low energy days because predicted cloud cover levels are more inaccurate when the weather is partly cloudy. The predictions become more accurate when the weather is mostly cloudy, as in the Very Low-Very Low days. Interestingly, while prediction accuracy drops as the prediction horizon stretches from 1 to 6 hours, beyond 6 hours accuracy sometimes improves. The reason is that the accuracy of the cloud cover information tends to vary widely with time. Of the 5 pairs of days we consider, predictions are most accurate for the High-High pair, with an average 1-hour ahead prediction error of 6.1%, and worst for the Low-Low pair, with an average 1-hour ahead prediction error of 23.8%.

To understand the impact of these mispredictions, we compare GreenVarPricesPeak when using our prediction vs. when using (idealized) perfect future knowledge of energy production. This comparison shows that the difference in green energy consumption, and total brown electricity cost per job between the two versions is always under 16.4%. The maximum difference occurs on the "Low-Low" days. This suggests that GreenHadoop could benefit somewhat from greater green energy prediction accuracy.

Scheduling for solar energy. Figures 5 and 6 show the behavior of Hadoop and EAHadoop, respectively, for the FaceD workload and the High-High days. The X-axis represents time, whereas the Y-axis represents cluster-wide power consumption (left) and brown energy prices (right). The figures depict the green and brown energy consumptions using areas colored light gray and dark gray, respectively. The two line curves represent the green energy available (labeled "Green actual") and the brown energy price ("Brown price").

These figures show that EAHadoop successfully reduces the overall energy consumption of the workload. This effect is most obvious around noon on Tuesday. However, both Hadoop versions waste a large amount of green energy (31% for both), which could be used instead of brown energy.

In contrast, Figure 7 depicts the behavior of GreenOnly under the same conditions. In this figure, we plot the amount of green energy that GreenHadoop predicted to be available an hour earlier (labeled "Green predicted"). The green prediction line does not exactly demarcate the light gray area, because our predictions are sometimes inaccurate.

A comparison between the three figures clearly illustrates how GreenOnly is capable of using substantially more green energy and less brown energy than Hadoop and EAHadoop, while meeting all job time bounds. GreenOnly spreads out job execution, always seeking to reduce the consumption of brown energy within resource and time constraints. Overall, GreenOnly consumes 30% more green energy than Hadoop and EAHadoop, respectively, in this experiment. Although GreenOnly does not consider brown energy prices, its brown electricity cost savings reach 30% and 29% compared to Hadoop and EAHadoop, respectively. (Note that these cost calculations do not consider peak brown power charges.)

Compared to Hadoop, the above gains come from: (1) batching of delayed jobs, which increases server energy efficiency and reduces overall energy consumption; (2) reducing idle energy by transitioning servers to low-power states; and (3) replacing some of the remaining brown energy with green energy. Compared to EAHadoop, the gains come from sources (1) and (3), as well as the fact that batching reduces the number of state transitions incurred by servers.

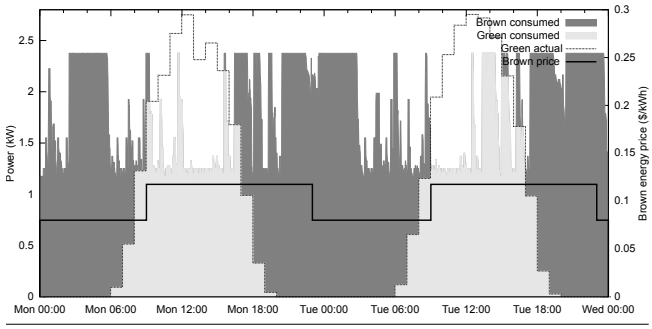

Figure 5. Hadoop for FaceD workload and High-High days.

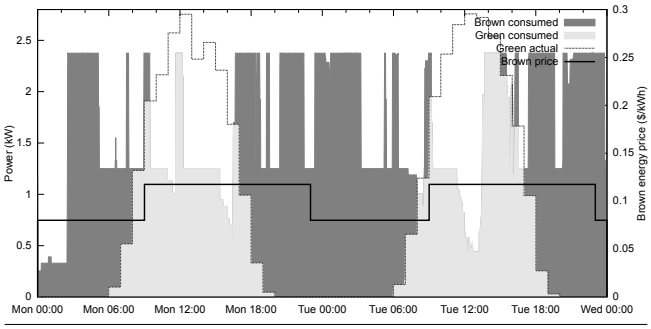

Figure 6. EAHadoop for FaceD workload and High-High days.

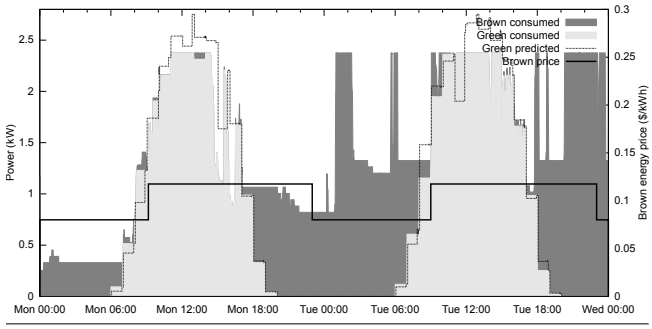

Figure 7. GreenOnly for FaceD workload and High-High days.

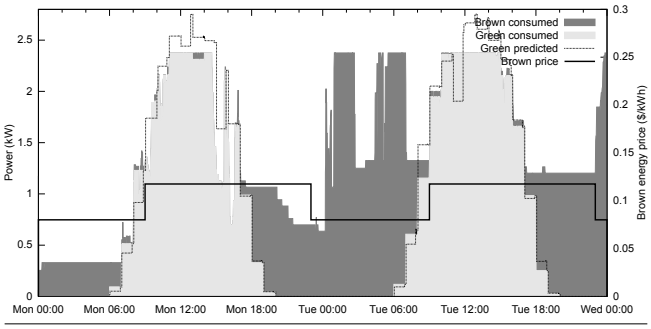

Figure 8. GreenVarPrices for FaceD and High-High days.

Scheduling for variable brown energy prices. GreenHadoop can reduce costs further when brown energy prices vary and brown energy must be used to avoid time bound violations. To quantify these additional savings, we now study GreenVarPrices in the absence of peak brown power charges.

Figure 8 shows the behavior of GreenVarPrices for FaceD and the High-High days. Comparing this figure against Figure 7, one can see that GreenVarPrices moves many jobs that must consume brown energy to periods with cheap brown energy. For example, GreenOnly runs many jobs on Tuesday night that consume expensive brown energy. Those jobs get scheduled during periods of cheap energy (starting at 11pm on Tuesday and lasting beyond the 2-day window depicted in the figure) under GreenVarPrices. As a result, GreenVarPrices exhibits higher brown electricity cost savings of 41% compared to Hadoop for the same days.

Scheduling for peak brown power charges. So far, we considered scenarios in which there are no peak brown power charges. However, datacenters are often subject to them [10]. Thus, we next study GreenVarPricesPeak and compare it to the other versions of GreenHadoop, Hadoop, and EAHadoop in the presence of those charges.

Figure 9 shows the behavior of GreenVarPricesPeak for FaceD and the High-High pair of days. Comparing this figure against Figure 8, one can see that GreenVarPricesPeak limits the peak brown power consumption as much as possible, while avoiding time bound violations. In this experiment, GreenVarPricesPeak reaches 1.47kW of peak brown power, whereas GreenVarPrices reaches 2.38kW. This lower peak translates into brown electricity cost savings of 39% and 37% compared to Hadoop and EAHadoop, both of which also reach 2.38kW.

To illustrate these behaviors further, Figure 10 shows the servers' states over time under GreenVarPricesPeak. The figure shows the number of servers that are in Decommissioned and Active states, as well as those active servers that are actually running jobs. The remaining servers are in Down state. The figure shows that GreenVarPricesPeak keeps the vast majority of the servers in Decommissioned or Down state, whenever green energy is unavailable. When it is available, GreenVarPricesPeak matches its availability by activating many servers to execute the load.

Figures 11 and 12 summarize our metrics of interest (with respect to the Hadoop results) in the absence and presence of peak brown power charges, respectively. Note that the cost savings bars represent the percent decrease in electricity cost per job executed. Considering the EAHadoop results, these figures demonstrate that it is not enough to consume less energy than Hadoop to increase the green energy consumption. To do so, it is also necessary to move the load around as in GreenHadoop. Considering GreenHadoop, Figure 11 clearly shows that its benefits are significant in terms of both green energy and brown electricity costs in the absence of peak brown power charges. When these charges are in effect (Figure 12), GreenOnly and GreenVarPrices achieve lower (but still important) savings in brown electricity costs, as they do not consider the charges in their scheduling. Both GreenOnly and GreenVarPrices reach the maximum peak brown power, so they incur the same peak brown power costs of Hadoop and EAHadoop. Overall, GreenVarPricesPeak is the

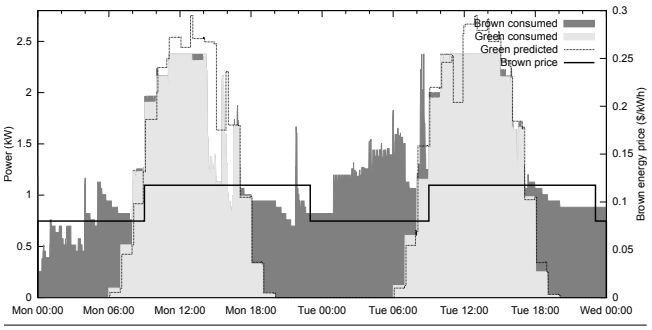

Figure 9. GreenVarPricesPeak for FaceD and High-High days.

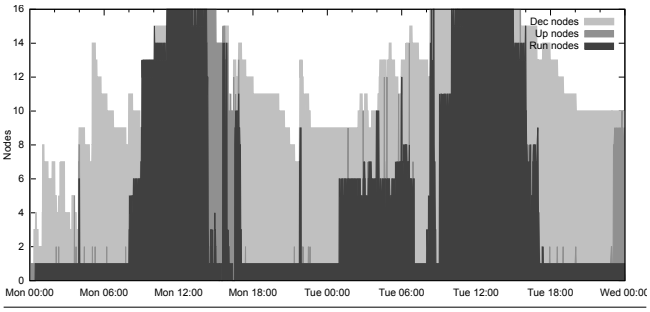

Figure 10. Server states under GreenVarPricesPeak. "Run servers" are active and running jobs. Down servers are not shown.

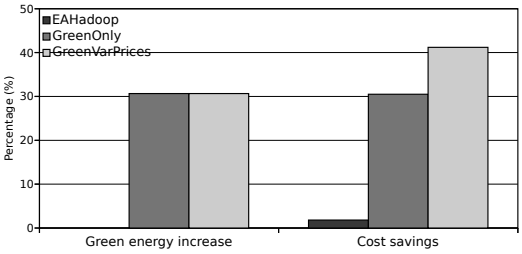

Figure 11. Summary in the absence of peak brown power charges.

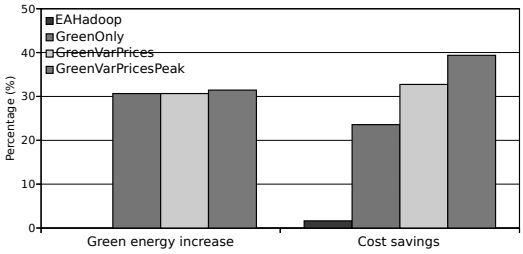

Figure 12. Summary with peak brown power charges.

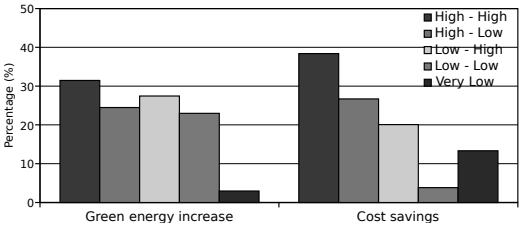

Figure 13. Impact of green energy availability on GreenVarPricesPeak. Results are normalized to EAHadoop.

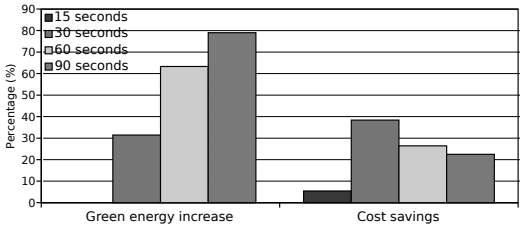

Figure 14. Impact of mean job inter-arrival time on GreenVarPricesPeak. Results are normalized to EAHadoop.

best version, as it explicitly and effectively manages the high costs of peak brown power consumption when those charges are in effect. When they are not, GreenVarPricesPeak reverts to GreenVarPrices.

Impact of the amount of green energy. Here we evaluate the behavior of GreenVarPricesPeak across the pairs of days we consider. Figure 13 summarizes the results with each bar in a group representing one pair of days. Since EAHadoop achieves roughly the same results as Hadoop but with a lower energy usage, we normalize against EAHadoop.

As one would expect, the figure shows that GreenHadoop increases green energy consumption by a smaller percentage when there is less green energy. However, its brown electricity cost savings remain high ($> 13\%$), except when there is almost no green energy *and* the prediction is inaccurate.

Impact of datacenter utilization. Another important factor that affects GreenHadoop is datacenter utilization. Under very high utilization, GreenHadoop may be unable to avoid using expensive brown electricity, may be forced to violate time bounds, and/or even reject newly submitted jobs.

To investigate these effects, we perform experiments with EAHadoop and GreenVarPricesPeak for three additional datacenter utilizations. Specifically, we vary the mean inter-arrival time from 15s (giving 92% utilization in EAHadoop) up to 90s (13% utilization). Recall that our default results assume a 30s mean inter-arrival time (49% utilization in EAHadoop). Figure 14 illustrates these results.

The figure shows that the environmental benefits of GreenHadoop tend to increase as utilization decreases. At 13% utilization, a common utilization in existing datacenters [25], GreenVarPricesPeak increases green energy use by 79%, and reduces brown electricity cost by 22%. At the uncommonly high utilization of 92%, there is little or no opportunity for GreenVarPricesPeak to increase green energy use. Utilization is so high that there is no need to move load around. However, GreenVarPricesPeak is still able to decrease brown electricity cost by 5% (by lowering the brown energy cost), while not violating any time bounds.

Impact of fraction of high-priority jobs. So far, we have assumed that all jobs in the workload are submitted with normal priority. Here, we investigate the impact of having dif-

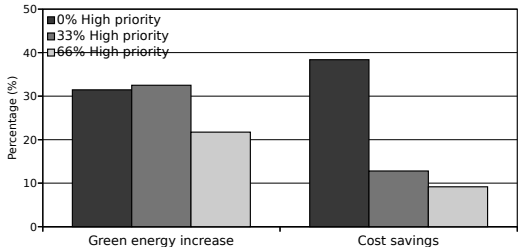

Figure 15. Impact of fraction of high-priority jobs on green energy and brown electricity cost. Normalized to EAHadoop.

ferent percentages of high-priority jobs. We selected jobs to be assigned high priority randomly, according to the desired percentage. Recall that GreenHadoop does not delay high- or very-high priority jobs.

Figure 15 shows the results for three percentages of high-priority jobs: 0% (our default), 33%, and 66%. As one would expect, larger fractions of high-priority jobs make it harder for GreenHadoop to increase green energy consumption and reduce brown electricity cost. Nevertheless, its benefits are still significant even at 66% high-priority jobs. Further, GreenHadoop achieves these benefits without degrading the performance of high-priority jobs noticeably.

Impact of shorter time bounds. The experiments so far use time bounds of one day. In those experiments, GreenHadoop never causes a time bound violation, nor does it need to take any action to prevent a time bound violation. We now explore GreenHadoop's ability to meet tighter time bounds of 6 and 12 hours. Our results show that GreenHadoop delays 19% of the jobs past their 6 hour time bounds (only 3% of the jobs are delayed by more than 20%), and 3% past their 12-hour time bounds. Note that some of the jobs in the FaceD workload take longer than 5 hours, making the 6-hour time bound tight. Further, most of the violations occur on the second day, after GreenHadoop has delayed many jobs to use green energy. At that point, an additional heavy load arrives, including two very large jobs, which prevents the system from meeting all bounds.

Impact of data availability approach. We compare GreenHadoop's data management against the covering subset approach [20]. Specifically, we run FaceD at different datacenter utilizations (from 13% to 92%) for the High-High days. The results show that, for low utilizations, GreenHadoop reduces brown electricity cost by more than 23% compared to the covering subset approach. GreenHadoop achieves this reduction by deactivating all but one server (when possible), whereas the covering subset approach requires at least four servers to be active at all times (see Section 3.1.2). Under high utilizations, the two approaches perform similarly because there are fewer opportunities to deactivate servers.

One potential drawback of our data management approach is that data may not be immediately available for arriving high-priority jobs. We run FaceD with 66% of high-priority jobs to assess this delay. The results show that all jobs start less than 6 seconds after submission; this delay can include the time to wake up to 15 servers.

Impact of workload (NutchI). Thus far, we have focused exclusively on FaceD. We have also studied NutchI to investigate whether GreenHadoop's benefits extrapolate to a substantially different workload. The results from these experiments support our most important observations with FaceD: (1) EAHadoop provides little benefit on its own; (2) GreenVarPricesPeak is able to conserve a significant amount of brown energy by batching load, transitioning servers to low-power states, and leveraging green energy; and (3) it increases the green energy consumption and reduces brown electricity cost significantly. For example, GreenHadoop increases green energy use by 23% and reduces brown electricity cost by 64% compared to Hadoop for the High-High days. These results show that GreenHadoop is robust to different workload characteristics.

Impact of energy estimation. Finally, an important component of GreenHadoop is its estimate of the average energy required per job in a batch of jobs. To study the accuracy of this estimation, we collected the estimates and actual averages over time for the experiment with GreenVarPricesPeak, FaceD, and High-High days. Note that jobs in FaceD are highly heterogeneous and so present a challenging scenario for our prediction. Approximately 75% of the predictions are within 20% of the observed average energy consumption. This shows that the estimate is reasonably accurate in most cases, but it can also be substantially inaccurate.

Our results above already provide strong evidence that GreenHadoop is robust to prediction inaccuracies. To further confirm this robustness, we evaluate how GreenHadoop reacts to a large, sudden change in the workload. Specifically, using the High-High days, we run FaceD during the first day and then abruptly switch to running NutchI on the second day. Right after this switch, GreenHadoop's energy usage prediction per job becomes off by approximately 50%. However, GreenHadoop is able to adjust its energy estimation to less than 15% error within two hours, and, at the end of the two days, uses 21% more green energy than EAHadoop, reducing the cost of brown energy by 57%.

GreenHadoop is robust to inaccurate energy usage predictions because of two reasons. First, the effect of inaccurate predictions is partly compensated by the much more numerous accurate ones. Second and most important, GreenHadoop re-evaluates and reacts to the system state every epoch. If it is over-estimating the average energy usage, it may be forced to use (more expensive) brown energy when not necessary. However, scheduled jobs will finish faster than expected, causing GreenHadoop to adjust by scheduling more jobs from the Wait queue. If it is under-estimating, in the worst case, it may end up causing jobs to miss their time bounds. However, pressure on the Wait queue will force GreenHadoop to schedule jobs faster, even if it has to start using (more expensive) brown energy.

Summary. The results above demonstrate that GreenHadoop provides consistent increases in green energy consumption (i.e., decreases in brown energy consumption) and reductions in brown electricity cost, compared to Hadoop and EAHadoop. The benefits are highest when most jobs are given low priority, green energy is abundant, and datacenter utilizations are low or moderate.

GreenHadoop may violate the time bound in some scenarios, including (1) the submission of very large jobs that are poorly described by the measured average energy used per job, (2) the arrival of a large burst of jobs after many previous jobs have been delayed to better use green energy, and/or (3) a very high datacenter utilization. However, our results show that GreenHadoop does not violate the time bound or does so only rarely with reasonable slack and datacenter utilization. For example, FaceD contains jobs that are up to 150 times longer than the common-case job. Yet, GreenHadoop never violates a 24-hour time bound and only rarely misses a 12-hour bound (3%).

6. Related Work

Exploiting green energy in datacenters. GreenHadoop lowers brown energy/power consumption, monetary costs, and environmental impact. Previous works have addressed some of these issues [17, 19, 22, 30]. Stewart and Shen discuss how to maximize green energy use in datacenters [30]. However, their main focus was on request distribution in multi-datacenter interactive services. Similarly, [17, 19, 22] focused on these services. Akoush *et al.* [1] considered workload distribution in virtualized systems. Our work differs in many ways. Specifically, only [22] considered green energy predictions, and only [17, 19, 22] considered variable brown electricity prices. None of these papers considered peak power or MapReduce job scheduling. MapReduce jobs typically run longer than interactive service requests and often have loose completion time requirements, thereby increasing the opportunity to exploit green energy.

Goiri *et al.* [9] and Krioukov *et al.* [14, 15] have proposed green energy-aware batch job schedulers for a single datacenter. Unlike GreenHadoop, however, these works require extensive user-provided information (numbers of required servers, run times, and completion time deadlines) for each job. Moreover, these schedulers do not manage data availability, assuming that the entire dataset of all jobs is either network-attached or replicated at every server. GreenHadoop is substantially more challenging, since it does not leverage any user-provided information about job behavior, and explicitly manages data availability.

Aksanli *et al.* [2] used green energy to process Hadoop jobs that share the same datacenter with an interactive service. However, they did not consider high-priority jobs, data management, brown energy prices, or peak power charges.

Willow [12] assumes that decreases in green energy supply affect the servers differently, and migrates load away from energy deficient servers within a datacenter. In contrast, Blink [27] considered managing server power states when the amount of green energy varies but the datacenter is *not* connected to the electrical grid. We argue that it is not realistic for datacenters to depend completely on green energy, since this may cause unbounded performance degradation. Our approach for managing green energy consumption is through scheduling, rather than load migration or server power state management.

At a much lower level, SolarCore [21] is a multi-core power management scheme designed to exploit PV solar energy. SolarCore focuses on a single server, so it is closer to the works that leverage green energy in embedded systems.

Managing brown energy prices and peak brown power charges. Most works that have considered variable energy prices have targeted request distribution across multiple datacenters in interactive Internet services [17, 19, 22, 24]. The exception is [18], which considers variable energy prices and peak power charges in multi-datacenter high-performance computing clouds. Our work differs in that it seeks to maximize green energy use, predict green energy availability, and schedule MapReduce jobs within a single datacenter.

Also within a single datacenter, Goiri *et al.* [9] scheduled scientific batch jobs taking into account brown energy prices, but not peak brown power charges. Govindan *et al.* [10] studied how energy stored in the UPS batteries of conventional datacenters can be used to manage peak power and its costs. GreenHadoop targets a different type of clustered system and relies solely on software to manage peak costs.

Traditional job schedulers. Traditional batch job schedulers for clusters, e.g. [8, 32], seek to minimize waiting times, makespan, and/or bounded slowdown; unlike GreenHadoop, they never consider green energy, brown energy prices, or peak brown power charges. In addition, similarly to real-time scheduling [23], GreenHadoop recognizes that many jobs have loose performance requirements (i.e., can be delayed within a bound) and exploits this in favor of higher green energy consumption and lower brown electricity cost.

MapReduce and Hadoop. Several efforts have sought to reduce the energy consumption of Hadoop clusters, e.g. [13, 20]. The main issue that these efforts address is how to place data replicas in HDFS, so that servers can be turned off without affecting data availability. Amur et al. addresses a similar issue in a power-proportional distributed file system, called Rabbit [3], based on HDFS. These data placement efforts could be combined with GreenHadoop to reduce the need for it to replicate data to turn servers off.

In contrast, Lang and Patel propose an approach called the All-In Strategy (AIS) [16]. Instead of turning some servers off when utilization is low, AIS either turns the entire cluster on or off. In essence, AIS attempts to concentrate load, possibly by delaying job execution, to have high utilization during on periods and zero energy consumption

during off periods. AIS's delay of load to increase efficiency is similar to our approach. However, AIS neither considers green energy availability nor brown energy and power costs.

7. Conclusions

In this paper, we proposed GreenHadoop, a MapReduce framework for a datacenter powered by solar energy and the electrical grid. GreenHadoop seeks to maximize the green energy consumption within the jobs' time bounds. If brown energy must be used, GreenHadoop selects times when brown energy is cheap, while also managing the cost of peak brown power consumption. Our results demonstrate that GreenHadoop can increase green energy consumption by up to 31% and decrease electricity cost by up to 39%, compared to Hadoop. Based on these positive results, we conclude that green datacenters and software that is aware of the key characteristics of both green and brown electricities can have an important role in building a more sustainable and cost-effective IT ecosystem.

To demonstrate this in practice, we are building a prototype micro-datacenter powered by a solar array and the electrical grid (`http://parasol.cs.rutgers.edu`). The micro-datacenter will use free cooling almost year-round and will be placed on the roof of our building.

Acknowledgements

We would like to thank Kevin Elphinstone and the anonymous reviewers for their help in improving our paper. We are also grateful to our sponsors: Spain's Ministry of Science and Technology and the European Union under contract TIN2007-60625, COST Action IC0804 and grant AP2008-0264, the Generalitat de Catalunya grants 2009-SGR-980 2010-PIV-155, NSF grant CSR-1117368, and the Rutgers Green Computing Initiative.

References

[1] S. Akoush et al. Free Lunch: Exploiting Renewable Energy for Computing. In *HotOS*, 2011.

[2] B. Aksanli et al. Utilizing Green Energy Prediction to Schedule Mixed Batch and Service Jobs in Data Centers. In *HotPower*, 2011.

[3] H. Amur et al. Robust and Flexible Power-Proportional Storage. In *SOCC*, June 2010.

[4] Apache. Apache Hadoop. http://hadoop.apache.org/, .

[5] Apache. Apache Nutch, . http://nutch.apache.org/.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, December 2004.

[7] DSIRE. Database of State Incentives for Renewables and Efficiency. http://www.dsireusa.org/.

[8] D. Feitelson et al. Parallel Job Scheduling – A Status Report. In *JSSPP*, June 2004.

[9] I. Goiri et al. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *Supercomputing*, November 2011.

[10] S. Govindan et al. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *ISCA*, June 2011.

[11] A. Jossen et al. Operation conditions of batteries in PV applications. *Solar Energy*, 76(6), 2004.

[12] K. Kant et al. Willow: A Control System for Energy and Thermal Adaptive Computing. In *IPDPS*, May 2011.

[13] R. T. Kaushik et al. Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System. In *CloudCom*, December 2010.

[14] A. Krioukov et al. Integrating Renewable Energy Using Data Analytics Systems: Challenges and Opportunities. *Bulletin of the IEEE Computer Society Technical Committee*, March 2011.

[15] A. Krioukov et al. Design and Evaluation of an Energy Agile Computing Cluster. Technical Report EECS-2012-13, University of California at Berkeley, January 2012.

[16] W. Lang and J. Patel. Energy Management for MapReduce Clusters. In *VLDB*, September 2010.

[17] K. Le et al. Cost- And Energy-Aware Load Distribution Across Data Centers. In *HotPower*, October 2009.

[18] K. Le et al. Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds. In *Supercomputing*, November 2011.

[19] K. Le et al. Capping the Brown Energy Consumption of Internet Services at Low Cost. In *IGCC*, August 2010.

[20] J. Leverich and C. Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. In *HotPower*, October 2009.

[21] C. Li et al. SolarCore: Solar Energy Driven Multi-core Architecture Power Management. In *HPCA*, February 2011.

[22] Z. Liu et al. Greening Geographical Load Balancing. In *SIGMETRICS*, June 2011.

[23] S. Oikawa and R. Rajkumar. Linux/RK: A Portable Resource Kernel in Linux. In *RTAS*, May 1998.

[24] A. Qureshi et al. Cutting the Electric Bill for Internet-Scale Systems. In *SIGCOMM*, August 2009.

[25] P. Ranganathan et al. Ensemble-level Power Management for Dense Blade Servers. In *ISCA*, June 2006.

[26] N. Sharma et al. Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems. In *SECON*, June 2010.

[27] N. Sharma et al. Blink: Managing Server Clusters on Intermittent Power. In *ASPLOS*, March 2011.

[28] SMA. Sunny Central 800CP, 2012.

[29] SolarBuzz. Marketbuzz, 2011.

[30] C. Stewart and K. Shen. Some Joules Are More Precious Than Others: Managing Renewable Energy in the Datacenter. In *HotPower*, October 2009.

[31] US Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency, August 2007.

[32] A. Yoo et al. SLURM: Simple Linux Utility for Resource Management. In *JSPP*, June 2003.

[33] M. Zaharia et al. Job Scheduling for Multi-User MapReduce Clusters. In *TR UCB/EECS-2009-55, Berkeley*, August 2009.