Matching Renewable Energy Supply and Demand in Green Datacenters[☆]

Íñigo Goiri[a], Md E. Haque[a,*], Kien Le[a], Ryan Beauchea[a], Thu D. Nguyen[a], Jordi Guitart[b], Jordi Torres[b], Ricardo Bianchini[a]

[a]*Department of Computer Science, Rutgers University, USA*
[b]*Universitat Politecnica de Catalunya and Barcelona Supercomputing Center, Spain*

Abstract

In this paper, we propose GreenSlot, a scheduler for parallel batch jobs in a datacenter powered by a photovoltaic solar array and the electrical grid (as a backup). GreenSlot predicts the amount of solar energy that will be available in the near future, and schedules the workload to maximize the green energy consumption while meeting the jobs' deadlines. If grid energy must be used to avoid deadline violations, the scheduler selects times when it is cheap. Evaluation results show that GreenSlot can increase solar energy consumption by up to 117% and decrease energy cost by up to 39%, compared to conventional schedulers, when scheduling three scientific workloads and a data processing workload. Based on these positive results, we conclude that green datacenters and green-energy-aware scheduling can have a significant role in building a more sustainable IT ecosystem.

Keywords: Green energy, energy-aware job scheduling, datacenters.

1. Introduction

Datacenters consume an enormous amount of energy: estimates for 2010 indicate that they consume around 1.5% of the total electricity used world-wide [1]. Electricity cost thus represents a significant burden for datacenter operators. Moreover, this electricity consumption contributes to climate change, since most of the electricity is produced by burning fossil fuels. A 2008 study estimated world-wide datacenters to emit 116 million metric tons of carbon, slightly more than the entire country of Nigeria [2]. We refer to the energy produced by carbon-intensive means and distributed via the electrical grid as "brown energy".

[☆]This submission is a modified and extended version of "GreenSlot: Scheduling Energy Consumption in Green Datacenters", which was originally published in SC'11.

[*]Corresponding author
Email address: mdhaque@cs.rutgers.edu (Md E. Haque)

These cost and environmental concerns have been prompting many "green" energy initiatives. One initiative is for datacenters to either generate their own renewable energy or draw power directly from a nearby renewable power plant. This approach is being implemented by many small and medium datacenters (partially or completely) powered by solar and/or wind energy all over the globe [3]. Larger companies are also investing in this direction. For example, Apple is building a 40MW solar array for its North Carolina datacenter [4]. McGraw-Hill has recently completed a 14MW solar array for its datacenter [5].

We expect that this trend will continue, as these technologies' capital costs keep decreasing (e.g., the inflation-adjusted cost of solar panels has decreased by 10-fold in the last three decades [6]) and governments continue to provide generous incentives for green power generation (e.g., federal and state incentives for solar power in the United States can reduce capital costs by up to 60% [7]). In fact, the trend may actually accelerate if carbon taxes and/or cap-and-trade schemes spread from Europe and Asia to the rest of the world. For example, a cap-and-trade scheme in the UK imposes caps on the brown energy consumption of large consumers [8]. We present a more extensive discussion of the feasibility of using green energy in datacenters in [9].

We argue that the ideal design for green datacenters connects them to both the solar/wind energy source and the electrical grid (as a backup). The major research challenge with solar and wind energy is that, differently from 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. Datacenters sometimes can "bank" green energy in batteries or on the grid itself (called net metering) to mitigate this variability. However, both batteries and net metering have problems: (1) batteries involve energy losses due to internal resistance and self-discharge; (2) the cost of purchasing and maintaining batteries can dominate in a solar system [9, 10]; (3) today's most popular battery technology for datacenters (lead-acid) uses chemicals that are harmful to the environment; (4) net metering incurs energy losses due to the voltage transformation involved in feeding the green energy into the grid; (5) net metering is not available 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.

Thus, in this paper, we investigate how to manage a datacenter's computational workload to match the green energy supply. In particular, we design a scheduler for parallel batch jobs, called GreenSlot, in a datacenter powered by an array of PV solar panels and the electrical grid. Jobs submitted to GreenSlot come with user-specified numbers of nodes, expected running times, and deadlines by which they shall have completed. The deadline information provides the flexibility that GreenSlot needs to manage energy consumption aggressively.

GreenSlot seeks to maximize the green energy consumption (or equivalently to minimize the brown energy consumption) while meeting the jobs' deadlines. If brown energy must be used to avoid deadline violations, it schedules jobs for times when brown energy is cheap. In more detail, GreenSlot combines solar energy prediction, energy-cost-awareness, and least slack time first (LSTF) job ordering [11]. It first predicts the amount of solar energy that will likely be available in the future, using historical data and weather forecasts. Based on its predictions and the information provided by users, it schedules the workload by creating resource reservations into the future. When a job's scheduled start time arrives, GreenSlot dispatches it for execution. Clearly, GreenSlot differs significantly from most job schedulers, which seek to reduce completion times or bounded slowdown.

We implement two versions of GreenSlot: one extends the SLURM scheduler for Linux [12], and the second extends the MapReduce scheduler of Hadoop [13]. We use real scientific workloads from the Life Sciences Department of the Barcelona Supercomputing Center to evaluate our SLURM extension and a Facebook-inspired workload to evaluate our Hadoop extension. Our results show that GreenSlot accurately predicts the amount of solar energy to become available. The results also show that GreenSlot can increase green energy consumption and decrease energy cost by up to 117% and 39%, respectively, for the workloads/systems evaluated.

Based on these positive results, we conclude that green datacenters and green-energy-aware scheduling can have a significant role in building a more sustainable Information Technology ecosystem.

In summary, we make the following contributions: (1) Introduce GreenSlot, a batch job scheduler for datacenters partly powered by solar energy; (2) Implement and evaluate GreenSlot in two different environments: a scientific computing cluster and a data-processing MapReduce cluster; and (3) Present extensive results isolating the impact of different aspects of the scheduler.

2. Background

Solar energy and datacenters. Solar is a promising clean energy technology, as it does not cause the environmental disruption of hydroelectric energy and does not have the waste storage problem of nuclear energy. Wind energy is also promising, but is not as abundant in many locations. Except for our (solar) energy predictions, our work is directly applicable to wind energy as well.

Transforming solar energy into (direct-current or DC) electricity is commonly done using PV panels. The panels are made of cells containing PV materials, such as monocrystalline and polycrystalline silicon. The photons of sunlight transfer energy to the electrons in the material. This energy causes the electrons to transfer between the two regions of the material, producing a current that is driven through the electrical load (e.g., a datacenter).

There are multiple ways to connect solar panels to a datacenter. Figure 1 shows an example. The AC Load is the server and cooling equipment, which typically runs on alternating-current (AC) electricity. The DC electricity is converted to AC using an inverter. Excess solar energy can be stored in batteries via a charge controller. The controller may also connect to the electrical grid, in case the datacenter must operate even when solar energy is not available. Where net metering is available, one can feed excess solar energy into the grid for a reduction in brown energy costs.

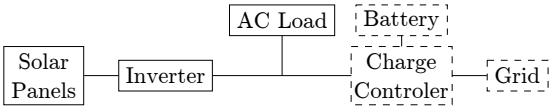

Figure 1: Components of a solar-powered system. Dashed boxes represent optional components.

The design we study in this paper does not include batteries or net metering, for the reasons we mentioned in the Introduction. We assume that the datacenter can be fully powered by the grid when insufficient green energy is being produced. On the other hand, any green energy that is not immediately used by the datacenter is wasted. Fortunately, GreenSlot is very successful at limiting waste. In fact, assuming the results from Section 5 and the best governmental incentives in the United States, the current capital cost of installing solar panels for the datacenter we model can be amortized by savings in brown energy cost in 10-11 years of operation. This period is substantially shorter than the 25+ years lifetime of the solar panels, and will be even shorter in the future, as solar costs continue to decrease at a rapid pace [6].

Brown energy prices. 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 energy consumed during an off-peak period than during an on-peak period. Typically, off-peak prices are in effect during the night, whereas on-peak prices apply during the day. Thus, it would be profitable for the datacenter to schedule part of its workload (e.g., maintenance or analytics tasks, activities with loose deadlines) during the night.

3. Related Work

Exploiting green energy in datacenters. GreenSlot schedules the use of green energy in datacenters to lower brown energy consumption, monetary costs, and environmental impact. Like GreenSlot, [9, 14–16] focused on managing batch jobs, whereas [17–20] considered interactive services or were not willing to delay computations. Batch jobs typically run longer than interactive service requests and often have loose deadlines, thereby increasing the opportunity to exploit green energy. GreenSlot differs from [14, 15, 21] as it considers both green energy and

brown energy prices in making its decisions. It differs from [15] in other important ways: [15] used only short-term green energy predictions and runs more or fewer batch jobs in arrival order as a function of green energy availability, without explicit deadlines; if green energy runs out, any started jobs are terminated. In contrast, GreenSlot schedules the jobs two days into the future, possibly reordering them, within their explicit deadlines. Jobs are never terminated, and may run completely on brown energy, if their deadlines so require.

GreenSlot differs from GreenHadoop [16] in that it leverages user-provided job run times, numbers of servers, and deadlines to schedule jobs more accurately. Liu *et al.* [22] focused on a similar problem as GreenSlot, but took a modeling and optimization approach to it.

To study real green datacenters, we have recently built Parasol, a small prototype datacenter powered by a solar array and the electrical grid [9]. We also built GreenSwitch, a software system for dynamically selecting the energy source, the medium for energy storage, and for scheduling deferrable and non-deferrable jobs [9]. GreenSwitch leverages some of the same ideas as GreenSlot for scheduling deferrable jobs, but targets datacenters with energy storage and does not rely on user-provided information about the jobs.

Other works [23–27] have considered green energy, but only in multi-datacenter setups. These works focus on workload distribution/migration, rather than on green energy-aware scheduling within each datacenter. Finally, [24, 28, 29] considered carbon offsetting as a different approach to greening datacenters.

Managing energy prices. Most of the works that have considered variable energy prices have targeted request distribution across multiple datacenters in interactive Internet services [23, 24, 26, 30]. GreenSlot differs from these efforts as it seeks to maximize green energy use, predict green energy availability, and schedule batch jobs within a single datacenter.

GreenSlot vs. conventional job schedulers. GreenSlot has a few unique characteristics, compared to other job schedulers, e.g. [12, 31]. First, it promotes the use of green energy and cheap brown energy, possibility at the cost of increasing job waiting times (but not violating deadlines). Talby and Feitelson [32] introduced the notion of increasing waiting times up to certain bounds in the context of backfilling. However, most job schedulers seek to minimize waiting times, makespan, and/or bounded slowdown; they never consider green energy or brown energy prices.

Second, GreenSlot borrows ideas from (soft) real-time systems: (1) jobs and/or workflows (i.e., sequences of related jobs [33]) have deadlines by which they shall complete; (2) it keeps the queued jobs in LSTF order [11]; and (3) new jobs that cannot be run before their deadlines are not admitted into the system. Although some previous job schedulers have considered deadlines (e.g., [34, 35]), most of them typically do not.

If the underlying scheduler (e.g., SLURM) allows job suspensions, GreenSlot suspends the jobs that outlast their allowed run times, instead of canceling them like most other schedulers do. As these jobs have already consumed energy, it would be wasteful to cancel them. The user can resume a suspended job although an expected remaining runtime must be given. GreenSlot will schedule the resumed job similar to a new job entering the system.

Run time estimates and deadlines. Prior research showed that users typically provide inaccurate estimates of run time [36, 37]. In fact, users often consciously overestimate to avoid job cancellations. Deadlines create another avenue for "gaming" the system; users may provide unnecessarily tight deadlines so that the scheduler executes their jobs ahead of others.

To alleviate these problems, we envision a computation pricing model for use with GreenSlot. To encourage users not to overestimate run times, users would pay in proportion to the actual run time of their jobs/workflows, but also pay a charge when they significantly overestimate those times. From this value, an amount proportional to how loose the deadlines are would be deducted. This model would achieve our two goals: tight expected run times, and loose deadlines. To compensate the user for a missed deadline, the datacenter operator would reimburse the user for an amount proportional to the length of the violation. Obviously, the payments in our model need not be in a real currency; rather, they could be effected in a virtual currency representing the right to use resources in the future, for example.

As another way of tackling poor run time estimates, GreenSlot could combine them with automatic predictions based on recent executions by the same users [38].

Hadoop. Some efforts have sought to reduce the energy consumption of Hadoop clusters. For example, [39, 40] focused on the careful placement of data replicas in Hadoop's distributed file system (HDFS), so that servers can be turned off without affecting data availability. These efforts can be combined with GreenSlot to reduce (or eliminate) the need for it to keep servers on only to serve data. In fact, our GreenSlot extension of Hadoop assumes the Covering Subset approach [39], under which one copy of the dataset is stored on the smallest possible number of servers; other servers can be deactivated without affecting data availability.

Lang and Patel proposed a different approach, called All-In Strategy (AIS), that turns the entire cluster on or off [41]. AIS attempts to concentrate load, possibly by delaying job execution, to have high utilization during the on periods. AIS considers neither the availability of green energy nor variable energy prices.

4. Scheduling in Green Datacenters

We propose GreenSlot, a parallel job scheduler for datacenters powered by PV solar panels and the grid. GreenSlot relies on predictions of the availability of solar energy, as well as on a greedy job scheduling algorithm.

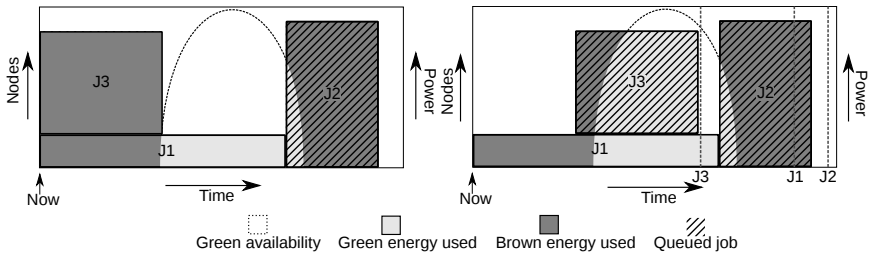

Figure 2: Scheduling 3 jobs (J1-J3) with backfilling (left) and GreenSlot (right). The jobs' deadlines are the vertical lines.

Figure 2 illustrates the behavior of GreenSlot (right), in comparison to a conventional EASY backfilling scheduler (left) for three jobs. Each rectangle represents the number of nodes and time that each job will likely require. The dashed vertical lines represent the jobs' deadlines. Note that backfilling uses less green energy (more brown energy), as it does not consider the energy supply in making decisions. Any scheduler (including a real-time one) that is unaware of green energy would behave similarly. In contrast, GreenSlot delays some jobs (within their deadlines) to guarantee that they will use green energy. This delay is not a concern since users only need their jobs completed by the jobs' deadlines. Similarly, GreenSlot may delay certain jobs to use cheaper brown energy (not shown). GreenSlot is beneficial because datacenters are not fully utilized at all times.

We next describe GreenSlot in detail. First, we describe our scheduling algorithm. Then, we present our model for solar energy prediction and discuss how GreenSlot adjusts the predictions when it finds inaccuracies.

4.1. Greedy Scheduling Algorithm

Overview. GreenSlot seeks to minimize brown energy consumption by instead using solar energy, while avoiding excessive performance degradation.

At submission, users can specify the workflows to which their jobs belong. As in many other job schedulers, users must specify the number of nodes and the expected running time for each job.Deadlines can be specified per job or workflow.

GreenSlot divides time into fixed-length "slots". At the beginning of each slot, GreenSlot determines if a new schedule must be prepared. If so, it goes through the list of queued jobs and schedules them (i.e., reserves resources for them) into the future. This scheduling window corresponds to the range of our hourly solar energy predictions, i.e. two days. The window is divided into smaller time slots (15 minutes in our experiments). The scheduling window moves with time; the first slot always represents the current time.

GreenSlot is cost-aware in that it favors scheduling jobs in time slots when energy is cheapest. To prioritize green energy over brown energy, green energy

is assumed to have zero cost. In contrast, brown energy prices often depend on time of use, as aforementioned. When the price is not fixed and brown energy must be used, GreenSlot favors the cheaper time slots. To avoid selecting slots that may cause deadline violations, GreenSlot assigns a high cost penalty to those slots. Any penalty that is large compared to the highest possible cost of a usable slot is appropriate.

GreenSlot is greedy in two ways: (1) it schedules jobs that are closer to violating their deadlines first; and (2) once it determines the best slots for a job, this reservation does not change (unless it decides to prepare a new schedule during a later scheduling round). The next job in the queue can only be scheduled on the remaining free slots. Moreover, GreenSlot constrains its scheduling decisions based on workflow information, i.e. a job belonging to phase i of a workflow cannot begin before all jobs of phases $< i$ have completed.

GreenSlot dispatches the jobs for execution, according to the schedule. Dispatched jobs run to completion on the same nodes where they start execution. GreenSlot deactivates any idle nodes to conserve energy.

Figure 3 illustrates GreenSlot's operation, from time T_1 (top) to T_3 (bottom), with a simple example. At T_1, job J1 is executing and job J2 is queued waiting for green energy to become available (according to GreenSlot's predictions). More than a day later than T_1, at T_2, J1 and J2 have completed, and J3 has just been dispatched. Because GreenSlot predicts two days of very little green energy, J4 is scheduled for the following day during a period of cheap brown energy. More than a day later than T_2, at T_3, we see that GreenSlot initially mispredicted the amount of green energy at time T_2. It later adjusted its prediction and ran J4 earlier. Finally, we also see J5 queued waiting for green energy to become available.

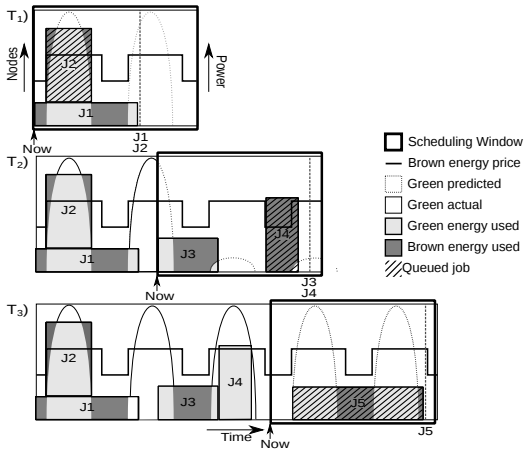

Figure 3: GreenSlot scheduling window at times T_1 (top), T_2 (middle), and T_3 (bottom).

Details. Figure 4 presents our scheduling algorithm. Line 0 lists the inputs that users must provide about each of their jobs and workflows. GreenSlot adds a small amount of tolerance (20% in our experiments) to each expected running time. If

the underlying scheduler allows job suspensions, jobs that take longer than this extended amount of time are suspended and must be re-started by hand (suspensions are not shown in Figure 4). Our goal is to tolerate some inaccuracy in the user-provided information, while avoiding deadline violations.

When a workflow has a deadline, GreenSlot creates tight internal deadlines for each of the phases of the workflow, based on the final deadline and the expected duration of the jobs (plus tolerance) in those phases. For example, consider a workflow with three phases that must be executed without overlap, each of which is expected to take 60 minutes. Suppose that the tolerance is 20%, i.e. the adjusted expected phase durations are 72 (60 + 12) minutes each. If the deadline for the workflow is 4pm, the internal deadlines for the first phase would be 4pm minus 144 minutes (1:36pm) and for the second phase 4pm minus 72 minutes (2:48pm).

Using the deadlines and the expected running times, GreenSlot determines the *latest possible start time* for each job. In the example above, the jobs of the first phase can start no later than 12:24pm, those of the second phase no later than 1:36pm, and those of the third phase no later than 2:48pm.

Lines 1-6 describe GreenSlot's behavior at the beginning of each time slot. It first determines whether its prediction for the amount of solar energy was accurate in the most recent slot (line 2). A prediction is deemed accurate if it had less than a 10% error (other reasonable thresholds produce similar results). If the predictions were inaccurate, GreenSlot adjusts the future predictions (line 3). We detail our approach to green energy prediction in the next subsection. If the predictions were adjusted, a new schedule must be prepared (line 4-5). A new schedule is also needed whenever a job arrives, a job completes, a job that was supposed to complete in the previous slot did not terminate, or there are jobs that were not scheduled in the previous scheduling round.

If a new schedule is needed, GreenSlot first subtracts the energy that the currently running jobs are likely to consume from the predicted amount of green energy for the scheduling window (line 8). (Currently, GreenSlot assumes that the administrator determines the average energy consumed by the jobs of each workflow based on their previous executions. We plan to automate this monitoring and integrate it into the scheduler.)

After updating the green energy availability, GreenSlot sorts the queued jobs in LSTF order. In more detail, it orders the queued jobs based on their remaining "slack", i.e. the difference between the current time and the latest possible start time (line 9). It then goes through the ordered list and schedules (reserves resources for) the jobs into the future (line 10-26). The key to scheduling each job is computing the energy cost of starting the job at each slot (lines 11-18). GreenSlot selects the starting slot that will lead to the lowest overall cost for the job (line 25), assuming that: (1) solar energy has zero cost; (2) the cost is infinite for any slot on

```
0. Users specify number of nodes, expected running time, deadline for each job/workflow
     Add tolerance to expected running times
1. At the beginning of each time slot:
2.   Determine whether the green energy predictions produced most recently were accurate
3.   If they were inaccurate: adjust the future predictions
4.   If predictions were just adjusted, a job arrived, a job completed,
         a job expected to complete on the previous slot did not, OR
         there are jobs to schedule:
5.     Prepare a new schedule
6.   Dispatch jobs according to schedule
7. Prepare schedule:
8.   Update the availability of green energy over time based on currently running jobs
9.   Try to schedule the next queued job in Least Slack Time First (LSTF) order
10.    Calculate cost of scheduling the job to start in each slot in the scheduling window
11.      The cost of starting the job on a slot should be infinite in the following cases:
12.      (1) a preceding job in the same workflow will not have completed until this slot
13.      (2) the job will end outside of the window
14.      (3) there are not enough nodes on this or at least one other needed slot
15.      When the cost is not infinite and brown energy is likely to be used:
16.        Account for the cost of the brown energy
17.      When the cost is not infinite, but the deadline will likely be violated:
18.        Add a violation penalty to the cost of the appropriate slots
19.    If cost is infinite for every slot:
20.      If job was submitted in this slot and deadline is within the window: reject it
21.      Otherwise: try to schedule this job in the next scheduling round
22.      Move to the next job (line 9)
23.    If a job would likely violate the deadline in every slot:
24.      Decrease its deadline (internally) by one slot
25.    Schedule job at the cheapest slot, except:
26.      A job with deadline outside the window should only be scheduled in the window
           if it can use green energy only (i.e., cost for cheapest slot = 0)
27.    Account for the energy and the nodes that will be used by the job
28.Dispatch jobs and adjust the number of active nodes:
29.  Activate nodes from S3 state, if necessary
30.  Start jobs that should be started now, according to the current schedule
31.  Send idle nodes to S3 state
```

Figure 4: GreenSlot algorithm. For simplicity, the pseudo-code assumes that no single job takes longer than the scheduling window. In addition, it does not show the suspension of jobs that have exceeded their expected running times (plus the tolerance).

which the job cannot start (lines 11-14); and (3) violating the deadline incurs an extra cost (lines 17-18). In computing costs, GreenSlot accounts for brown energy prices (line 16). Importantly, it requires no modifications to tackle scenarios in which the brown energy price is fixed.

When multiple slots would lead to the same lowest overall cost, GreenSlot selects the earliest of the tied slots for the job, if the lowest cost is zero (only green energy would be used). When there is a tie but the lowest cost is not zero, GreenSlot selects the latest of the tied slots but only if there is a chance that more green energy may become available (due to a misprediction) until then. In this case, when the prediction is corrected, GreenSlot can move the job back earlier in the schedule so that it uses all the green energy that is really available. If instead GreenSlot

overestimated the amount of green energy, it will still use all the available green energy. However, it might have to resort to using expensive brown energy for some of the jobs that were delayed.

A new job with deadline within the current window that cannot be scheduled on any slot of the window is not admitted into the system (line 20). This behavior allows the user to re-submit the job with a later deadline or fewer nodes. Any other job that cannot be scheduled is simply put back on the queue; the job has already been admitted into the system, so GreenSlot cannot reject it any more. GreenSlot will try to schedule it in the next scheduling round (line 21). Similarly, GreenSlot leaves any job with a deadline beyond the current window for the following scheduling rounds, unless it predicts to have enough green energy to execute it within the current window (line 26).

GreenSlot treats jobs that are expected to take longer to execute than the length of the time window differently (not shown in Figure 4 for clarity). These jobs are scheduled as soon as resources allow.

Because GreenSlot is greedy and only sees a finite amount of time into the future, it may be unable to prevent deadline violations by leaving too many jobs to be executed beyond its horizon. It mitigates this problem by internally decreasing by one slot the deadline of any job expected to miss its deadline according to the schedule (line 23-24). The earlier deadline decreases the job's slack time. As a result, the next time the schedule is prepared, this job will have a greater chance of being scheduled before the jobs that are preventing it from meeting its deadline.

Finally, lines 28-31 implement GreenSlot's job dispatcher. The dispatcher is mainly tasked with starting the jobs scheduled to start on the current time slot (the first slot of the window). Before doing so, the dispatcher may need to activate nodes that it earlier transitioned to ACPI's S3 state (also known as suspend-to-RAM state). This state consumes very low power (8.6 Watts in our machines) and can be transitioned to and from quickly (7 seconds total in our machines). Because of these fast transitions, the dispatcher sends any idle nodes to S3 state instead of turning them completely off. Turning nodes off would involve transition times of multiple minutes, which would represent a significant overhead compared to the length of GreenSlot's time slots.

Limitations. GreenSlot may potentially reject more jobs or miss more deadlines than a scheduler that delays fewer jobs. However, as our sensitivity study in Section 5.3.1 shows, this is only likely to occur in datacenters with unusually high utilizations. In fact, we have not seen any job rejections or missed deadlines under the more common (yet still relatively high) utilizations and real workloads we study. A full evaluation of these effects is a topic for our future work.

4.2. Predicting the Availability of Solar Energy

Our model for predicting the generation of solar energy is based on a simple premise: various weather conditions, e.g., partly cloudy, reduce the energy generated in a predictable manner from that generated on an ideal sunny day. This premise is expressed as $E_p(t) = f(w(t))B(t)$, where $E_p(t)$ is the amount of energy predicted for time t, $w(t)$ is the weather forecast, $f(w(t))$ is a weather-dependent attenuation factor (between 0 and 1), and $B(t)$ is the amount of energy expected under ideal conditions.

We implement solar energy prediction using the above model at the granularity of an hour. We use weather forecasts available from sites such as The Weather Channel to instantiate $w(t)$. These sites provide hourly predictions for up to 48 hours into the future (which explains why the scheduling window is two days). Each prediction includes a string describing the forecasted condition such as "cloudy" or "scattered thunderstorms". This string is the output of $w(t)$.

We use historical data to instantiate both $B(t)$ and $f(w(t))$. Specifically, for a given hour t, we use the actual weather conditions and energy generated during the month centered on t from the previous year. We choose this "reference" month around t to account for seasonal effects. We set $B(t)$ to the maximum energy generated for the same hour of any day in the reference month. For each weather condition wc, we compute $f(wc)$ as the median amount by which wc decreased $B(t)$ whenever this condition was reported during the reference month. Note that $f(wc)$ is always between 0 and 1 since $B(t)$ is the maximum observed energy generated for the same hour in the day of the reference month.

Unfortunately, weather forecasts can be wrong. For example, we have observed that thunderstorm forecasts are frequently inaccurate and can remain inaccurate throughout a day; i.e., the forecast continues to predict a thunderstorm hour-by-hour but the storm never arrives. Further, 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 during the above "mispredictions", we also use an alternate method of instantiating the attenuation factor for time t. Specifically, we assume that the recent past can predict the near future, and compute this factor using the observed energy generated in the previous hour. When invoked, our prediction module compares the accuracy of the two methods for predicting the energy generated during the last hour, and chooses the more accurate method to instantiate the attenuation factor for the remainder of the current day. Beyond the current day, we always instantiate this factor using weather forecasts because weather conditions can change significantly from one day to the next.

Although we do not claim our prediction approach as a contribution of this paper, it does have three important characteristics: it is simple, relies on widely

available data, and is accurate at medium time scales, e.g. a few hours to a few days. Previous works have proposed more complex models based on historical weather data [42]. However, these models tend to be inaccurate at medium time scales [43]. Based on this observation, Sharma *et al.* proposed a simple model based on historical data and weather forecasts [43]. Our approach is similar, but also embodies error correction based on the recent green energy production.

4.3. GreenSlot Implementations

We built two implementations of GreenSlot: the first extends the SLURM parallel job scheduler for Linux, and the second extends the MapReduce scheduler of Hadoop. The core of GreenSlot consists of 2300 uncommented lines of Python code that are independent of the underlying scheduler. The first implementation adds another 500 uncommented lines of SLURM-related Python code for a total of 2800 lines. The second implementation consists of 60 uncommented lines of Java code to make Hadoop energy-aware and another 200 lines of Hadoop-related Python code. In the absence of GreenSlot, both SLURM and Hadoop schedule jobs in First-Come First-Served fashion without any delays.

5. Evaluation

5.1. Methodology

Hardware and software. We evaluate GreenSlot using a 16-node cluster, where each node is a 4-core Xeon server with 8GB of memory, 1 7200rpm SATA disk, and a 1Gb/s Ethernet card. GreenSlot runs on an additional server. The servers are connected by a Gigabit Ethernet switch. We measure power with an accurate Yokogawa multimeter. Our servers consume up to roughly 150W, whereas the switch consumes 55W and the low-power server that runs GreenSlot consumes roughly 30W.

Solar panel array. We model the solar panel array as a scaled-down version of the Rutgers solar farm. The farm can produce 1.4MW of power (after DC to AC conversion) that is used by the entire campus. By computing the actual energy production over time with respect to this maximum power, we can estimate the production of smaller installations. In particular, we scale the farm's AC production down to 10 solar panels capable of producing 2.3kW of power. We selected this scaled size because, after conversion, it produces roughly the common-case peak power consumption of our system.

We considered one year worth of solar energy production by the farm, from March 8th 2010 to March 7th 2011. The scaled-down daily productions for the weekdays in this period can be found in `http://www.darklab.rutgers.edu/`

`GreenDC/solar.html`. We collected weather forecast data for 30 of these weeks. From this set, we picked 4 weeks to study in detail: the week with the most solar energy (starting on May 31th 2010), the week with the average amount of solar energy (starting on July 12th 2010), a week with little solar energy in the first three days but later significant energy (starting on August 23rd 2010), and a week with lots of solar energy in the first two days but later little solar energy (starting on March 7th 2010). We call these weeks "Most", "Average", "Low-High", and "High-Low", respectively.

Cost and brown energy prices. We consider the electricity cost required to complete a given workload. This cost is computed assuming the most common type of variable pricing for brown (grid) energy, namely on-peak/off-peak pricing. In on-peak/off-peak pricing, brown energy costs less when used during off-peak consumption times (from 11pm to 9am) and more when consumed during on-peak times (from 9am until 11pm). The difference between on-peak and off-peak prices is largest in the summer time (June-September). We assume the 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). Summer prices apply to the Most, Average, and Low-High weeks. We assume that the operational cost for generating solar energy is 0 (although as discussed in the Introduction, there is a capital cost that has to be amortizable for the solar power plant to result in a net cost saving over its lifetime).

Accelerating and validating the experiments. It would be impossible to perform all of the experiments in this paper in real time. This would require hundreds of days of non-stop experiments. To speed up our study, we accelerate the experiments by a factor of 100. This means that a job that takes 100 minutes in real time completes in just 1 minute in the accelerated experiment. In addition, it means that five days of real time elapse in 72 minutes.

To verify that an accelerated run is faithful to its real-time counterpart, we run a validation experiment for 31 hours — from Monday at 9am until Tuesday at 4pm — with GreenSlot for SLURM scheduling our real scientific computing workloads (described in Section 5.3 below) with their estimated run times and deadlines. The corresponding accelerated run shortens all job-related times by 100x. Specifically, the accelerated jobs do not perform actual work; they simply occupy the nodes for the proper amount of time. Both runs assume on-peak/on-peak brown prices. GreenSlot itself cannot be accelerated. In this experiment, it takes a maximum of 0.3 seconds (without any optimizations) to prepare a full schedule on an Intel Atom-based server. This maximum occurs when the largest number of jobs (70) is in the queue. As Figure 4 suggests, GreenSlot's execution time is proportional to the number of jobs in the system.

The validation results demonstrate that the accelerated runs are very accurate.

	Prediction Error (%)					
	1	**3**	**6**	**12**	**24**	**48**
Median	12.9	15.6	15.8	16.1	16.5	19.0
90^{th} %	24.6	33.9	40.5	44.1	42.5	44.4

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

In detail, the real-time and accelerated runs differ by at most 2.3% with respect to the 4 metrics of interest: amount of green energy used (difference of 0.7%), amount of brown energy used (2.3%), energy cost (1.9%), and number of deadlines violated (no violations in either run).

5.2. Solar Energy Predictions

We evaluate our solar energy predictor using data collected from the Rutgers solar farm, scaled as described above, and weather.com (actual and predicted conditions) for seven months: June–September 2010 and January–March 2011. Table 1 shows the normalized percentage prediction error for daily energy production when predicting 1 to 48 hours ahead. 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., $\sum_{t=0}^{23} B(t)$). When predicting x hours ahead, we use the weather forecast obtained at time $t - x$ to predict production at time t.

These results show that our predictor is reasonably accurate, achieving median and 90^{th} percentile errors of 12.9% and 24.6%, respectively, when predicting energy production for the next hour. That is, 50% of the time, our predictions across the hours of a day is off by 12.9% or less of the daily generation capacity (~14.8kWh). Further, though accuracy degrades with prediction horizon, this degradation is small beyond 3 hours. Even when predicting 48 hours ahead, the median error is 19.0%.

Of the 4 weeks we use, week Low-High has the best prediction accuracy, with a median 1-hour ahead prediction error of 9.3%, while week High-Low has the worst prediction accuracy, with a median 1-hour ahead prediction error of 18.4%. The other two weeks have errors close to the ones listed above.

5.3. GreenSlot for SLURM

GreenSlot variations and baseline for comparison. We study two variations of the SLURM-based version of GreenSlot: "GreenOnly", which considers green energy availability, but not variable brown energy prices; and "GreenVarPrices", which considers both green energy and variable brown energy prices.

For comparison, we study a variant of EASY backfilling [44] that considers the deadlines in sorting the job queue in LSTF order. The scheduler backfills jobs, as long as the first job in the queue is not delayed. We refer to this scheduler as

"Conventional". Like GreenSlot, Conventional assigns a 20% tolerance to the user-estimated run times. If a job's estimate and tolerance are exceeded, Conventional cancels the job. It transitions unneeded servers to ACPI's S3 state to save energy.

Workloads. We use 3 scientific computing workloads in production use at the Life Sciences Department of the Barcelona Supercomputing Center [45]. Each workload implements a different pipelined approach to the sequencing and mining of the genome of a baker's yeast. Each workload runs for 5 days and comprises a set of workflows, each of which analyzes a different yeast sample. Workload1 and Workload3 have 8 workflows each, whereas Workload2 has 12 workflows. Each workflow of Workload1 comprises 4 phases: initialization (1 job that runs for 8 minutes on our cluster), data splitting (1 job that runs for 1 minute), computation (16 jobs that last between 6 minutes and 9 hours, with an average of 2.4 hours), and collect/visualization (1 job that runs for 5 minutes). Each workflow of Workload2 comprises 3 phases: initialization and splitting (1 job that runs for 10 minutes), computation (8 jobs that last between 2 hours and 9 hours, with an average of 4 hours), and collect/visualization (1 job that runs for 5 minutes). Each workflow of Workload3 also comprises 3 phases: initialization and splitting (1 job that runs for 10 minutes), computation (8 jobs that last between 1.25 hours and 2.27 hours, with an average of 1.26 hours), and collect/visualization (1 job that runs for 5 minutes). In total, there are 352 jobs and 28 workflows in these workloads. On average, the input data for each workflow is 1.2 GB, the intermediate file sizes are 800 MB each, and the final output size is 100 MB. Our Life Sciences colleagues run these workloads on a cluster of the same size as our own, so we do not scale them.

Starting on Monday at 9:30am of every week, a workflow from each workload is submitted every 30 minutes. The workflows of Workload1 and Workload3 have deadlines every day at 9:00am and 2:00pm from Tuesday until Friday. The workflows of Workload2 have deadlines every day at 9:00am, 1:00pm, and 4:00pm from Tuesday until Friday. The reason for the staggered deadlines is that they give the researchers time to interpret the results before they are shipped to another research group. Since our workloads run from Monday to Friday, we loosely refer to these five days as a week. This configuration corresponds to approximately 50% cluster utilization, which is comparable to (or even higher than) many real scientific-computing datacenters and grids [46, 47].

As it is clear from the description above, the computation jobs represent the vast majority of the jobs and the time in the workloads. These are multithreaded jobs that use as many cores as are available at the server on which they run. There are no multi-node jobs in the real workloads. In Section 5.4, we evaluate a workload with multi-node jobs that arrive over time, rather than clustered on Monday. Finally, our experiments assume that the user-provided estimates of job run time

are exactly the run times listed above. We have studied the impact of inaccuracies in runtime estimates of up to [-40%,+20%], and our results (not shown here because of space constraints) show that such inaccuracies have essentially negligible impact on GreenSlot.

Power consumption. We measured the power consumption of each job in each workflow. The computation jobs almost constantly consume 105W, whereas the initialization jobs consume 140W, the splitting jobs consume 90W, and the collection/visualization jobs consume 102W. Overall, the common-case peak power consumption for our scientific workloads is 1765W = 16 × 105W + 55W (switch) + 30W (GreenSlot). A server consumes 8.6W in the S3 state. Transitioning into and out of S3 takes 7 seconds. When scheduled by the Conventional scheduler, the week-long workload consumes 75.38kWh on 16 nodes.

5.3.1. Results

This section presents our experimental results. First, we isolate the impact of being aware of green energy by comparing GreenOnly with Conventional. These results also assess the impact of the quality of green energy predictions on our scheduling. Second, we study GreenVarPrices to isolate the benefit of being aware of brown energy prices. Third, we study the impact of the datacenter utilization on GreenVarPrices. Finally, we quantify the impact of poor run time estimates.

In our experiments, Conventional and GreenSlot do not violate any deadlines, except when we explore high datacenter utilizations to purposely cause violations.

Scheduling for solar energy and impact of predictions. Figure 5 shows the behavior of Conventional for our workloads, the Average week, and accurate job run time estimates. The X-axis represents time, whereas the Y-axis represents cluster-wide power consumption (left) and brown energy prices (right). The figure depicts the green and brown energy consumptions in 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").

As Conventional schedules the workloads to complete as soon as possible, it heavily uses the servers early in the week and leave them in deep-sleep state late in the week. This approach is ideal in terms of conserving energy, since keeping modern servers powered on involves a high "static" energy. However, Conventional wastes a large amount of green energy, which could be used instead of brown energy. In this experiment, only 26% of the energy consumed is green.

Figure 6 depicts the behavior of GreenOnly, under the same conditions as in Figure 5. In this figure, we plot the amount of green energy that GreenSlot 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

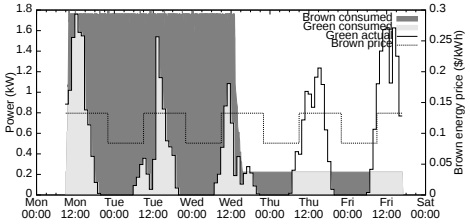

Figure 5: Conventional scheduler and Average week.

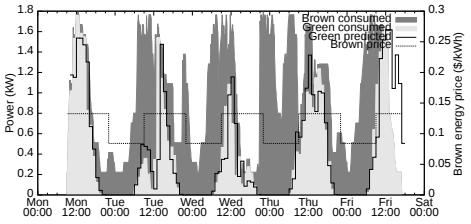

Figure 6: GreenOnly scheduler and Average week.

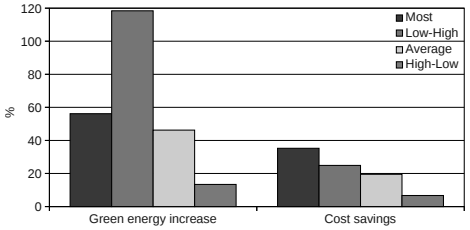

Figure 7: GreenOnly's green energy increase and cost savings.

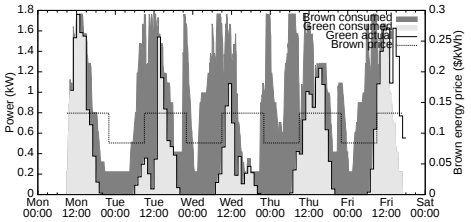

Figure 8: GreenOnly with actual green energy availability.

sometimes do not match the actual green energy available.

A comparison between Figures 5 and 6 clearly illustrates how GreenOnly is capable of using substantially more green energy than Conventional, while meeting all job/workflow deadlines. GreenOnly spreads out job execution across the week, always seeking to reduce the consumption of brown energy within resource and deadline constraints. Overall, GreenOnly consumes 47% more green energy than Conventional in this experiment. Although GreenOnly does not explicitly consider brown energy prices in making decisions, its energy cost savings reach 20% compared to Conventional. More than 80% of these cost savings comes from replacing brown energy with green energy.

The results for the other weeks are similar, as seen in Figure 7. The figure shows two sets of 4 bars. The set on the left represents the increase in green energy consumption, whereas the set on the right represents the energy cost savings. Each bar represents a week. Overall, GreenOnly increases green energy consumption between 13% and 118%, and reduces costs between 7% and 35%. GreenOnly improves on Conventional even for the worst-case week (High-Low) for us.

Another interesting observation is that our predictions of green energy availability are plenty accurate for our purposes. The green availability curve traces the gray area in Figure 6 well. To quantify the impact of prediction accuracy, consider Figure 8. The figure shows the behavior of GreenOnly under the same conditions, except that we use the actual green energy availability (representing idealized perfect knowledge of future energy production) instead of our predictions of it. A comparison of Figures 6 and 8 shows similar schedules. Overall, we find that per-

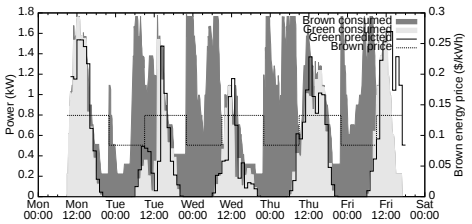

Figure 9: GreenVarPrices and Average week.

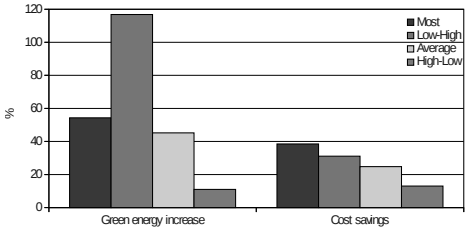

Figure 10: GreenVarPrices' green energy increase and savings.

fect knowledge increases green energy use and decreases cost both by only 1%. Thus, this experiment is the only one in which we consider perfect knowledge of green energy availability.

Scheduling for solar energy and brown energy prices. So far, we have studied scheduling that does not explicitly exploit variable brown energy prices. However, GreenSlot can reduce costs further when brown energy prices vary and brown energy must be consumed to avoid deadline violations. To quantify these savings, we now consider the GreenVarPrices version of GreenSlot.

Figure 9 shows the behavior of GreenVarPrices again for our real workloads, the Average week, and accurate job run time estimates. Comparing this figure against Figure 6, one can clearly 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, Wednesday night, and Thursday night that consume expensive brown energy. Those jobs get scheduled during periods of cheap energy under GreenVarPrices. As a result, GreenVarPrices exhibits higher energy cost savings of 25% compared to Conventional for this week, while consuming almost the same amount of green energy as GreenOnly.

GreenVarPrices achieves positive results for the other weeks as well, as illustrated in Figure 10. Overall, the GreenVarPrices cost savings range from 13% to 39%, whereas its increases in green energy consumption range from 11% to 117%.

A comparison between Figures 7 and 10 illustrates the benefit of considering brown energy prices explicitly in GreenSlot. As one would expect, doing so decreases costs with respect to GreenOnly. To isolate GreenSlot's ability to exploit cheap brown energy in the absence of green energy, we also consider an idealized week with no solar energy. For this week, GreenVarPrices reduces energy cost by 13% with respect to Conventional.

Impact of datacenter utilization. Another important factor in evaluating GreenSlot is its behavior as a function of datacenter utilization. Under high enough utilization, GreenSlot may unable to avoid using expensive brown energy, may be forced to violate deadlines, and/or even cancel newly submitted jobs.

To investigate these effects, we perform experiments with Conventional and GreenVarPrices for four additional datacenter utilizations: 67%, 72%, 87%, and 92%. We achieve these higher utilizations by adding four, five, eight, and nine extra copies of Workload3, respectively. Recall that our other experiments utilize the datacenter at 50%, which is already a relatively high utilization in many scientific environments [46, 47].

These results show that GreenVarPrices does not start violating deadlines until the utilization reaches an uncommon 72%. At 67% utilization, GreenVarPrices still increases green energy consumption by 31% and reduces energy cost by 14% in comparison to Conventional at the same utilization. In contrast, Conventional only starts violating deadlines at 92% utilization.

Although one could concoct scenarios that would challenge GreenSlot to a greater extent, these results with real workloads suggest that GreenSlot is robust to high but still realistic utilizations. Moreover, a higher level scheduler could easily select between GreenSlot or Conventional based on the current utilization.

5.4. GreenSlot for Hadoop

The results thus far used real workloads. However, one may argue that these workloads favor GreenSlot, in that there are no multi-node jobs and all workflows are submitted on Monday. To show that GreenSlot is robust to different environments and different workload characteristics, in this section, we evaluate GreenSlot as implemented for Hadoop. Besides the different underlying scheduler, this evaluation involves jobs that run on multiple servers, and arrive throughout the week.

GreenSlot variations and baseline. We first modify Hadoop to allow jobs to be submitted with the number of nodes that they should run on, and to ensure that each job is scheduled on no more than the specified number of nodes. In addition, we modify Hadoop such that any unneeded servers outside the Covering Subset are transitioned to ACPI's S3 state to save energy. We call this system "EAHadoop" (short for Energy-Aware Hadoop), and use it as the baseline for comparison.

We built GreenSlot as an extension of EAHadoop. We again call the full implementation "GreenVarPrices". In this system, each MapReduce job is represented by a simple workflow comprising a map phase and a reduce phase. Each Hadoop node is configured to have m map spots and r reduce spots, i.e. a node can simultaneously run m map tasks and r reduce tasks. GreenSlot must schedule map tasks only in the map spots and reduce tasks only in the reduce spots. The user-provided number of nodes n is multiplied by m to get the maximum number of map tasks that can be run simultaneously, and multiplied by r to get the total number of reduce tasks. In our experiments, $m = 4$ and $r = 2$.

Workload. Our data-processing workload is modeled after the Facebook workload

	Map Tasks				
Reduce Tasks	**0**	**1**	**2**	**3**	**4**
0	62.0	79.2	87.1	91.3	95.0
1	82.0	81.3	84.3	91.3	99.5
2	94.4	87.7	84.1	97.6	103.9

Table 2: Power (in Watts) vs. number of map and reduce tasks.

described in [48], but simplified (by consolidating 9 groups of different job sizes into 3 groups) and scaled down for our smaller cluster. It consists of 75% small, 13% medium, and 12% large jobs. Each of the jobs is a TeraSort application [49], a common Hadoop benchmark. Each small job comprises 20 map tasks and 10 reduce tasks, runs on 5 nodes,[1] and takes 2.8 hours on average. Each medium job comprises 40 map tasks and 20 reduce tasks, runs on 7 nodes, and takes 4.5 hours on average. Each large job comprises 80 map tasks and 40 reduce tasks, runs on 11 nodes, and takes 5.4 hours on average. Job arrival follows a Poisson distribution with an average inter-arrival time of ~66.7 minutes, corresponding to approximately 50% cluster utilization. Other researchers have assumed Poisson arrivals for Hadoop [50]. Deadlines are 6 hours, 12 hours, and 24 hours for small, medium, and large jobs, respectively. Recall that we accelerate all job run times and deadlines.

Power consumption. As Table 2 shows, the per-node power consumption depends on the number of map and reduce tasks currently running on the node. When a node is kept active just to provide data, i.e., 0 map and 0 reduce tasks, it consumes approximately 62W. Overall, the week-long workload consumes 100.27kWh on 16 nodes, when scheduled by EAHadoop.

5.4.1. Results

We compare the behaviors of GreenVarPrices and EAHadoop for the same 4 weeks as before. Again, neither system violated any deadlines.

Figures 11–14 show the behaviors of EAHadoop and GreenVarPrices for the Most and Average weeks. Figure 15 plots the increase in green energy usage and cost savings for GreenVarPrices compared to EAHadoop.

Overall, GreenVarPrices achieves cost savings of 28–31%, and increases green energy consumption by 19–21%. Figures 11–14 show that the workload peaks can be misaligned with green energy production when jobs are executed immediately on their arrivals. Also, jobs may be executed during periods of high energy prices. GreenVarPrices achieves its cost savings and increases in green energy usage by

[1] We carefully chose the number of nodes per job to create fragmentation in our 16-node cluster. For example, 2 small jobs and 1 medium job cannot be scheduled simultaneously because they require 17 nodes. This makes it harder for GreenSlot to maximize green energy usage.

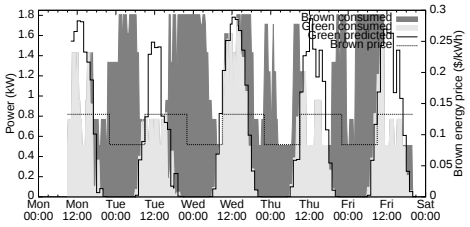

Figure 11: EAHadoop and Most week.

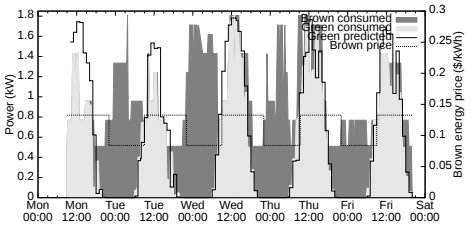

Figure 12: GreenVarPrices for EAHadoop and Most week.

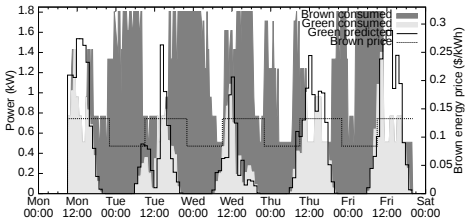

Figure 13: EAHadoop and Average week.

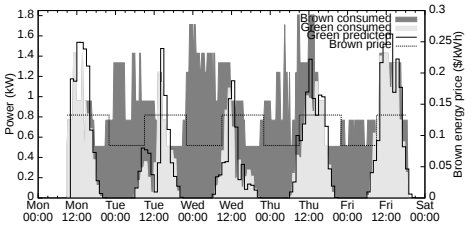

Figure 14: GreenVarPrices for EAHadoop and Average week.

delaying jobs to execute during periods of high green energy production or low brown energy prices.

These experiments exhibit smaller benefits than GreenSlot for SLURM for two reasons. First, some of the nodes (the Covering Subset) must be kept on all the time to ensure data availability. Thus, some green energy is always consumed by these nodes. Second, the workload is more spread out throughout the week, so that entire periods of green energy production (end of the week) are not missed as before. This characteristic of the workload is also the reason for all the weeks to exhibit similar results. Despite these factors, the cost savings and increases in green energy consumption remain substantial, showing that GreenSlot is robust to different underlying schedulers, implementations, and workload characteristics.

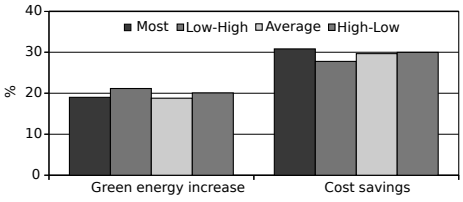

Figure 15: GreenVarPrices' for EAHadoop green energy increase and cost savings.

6. Conclusions

In this paper, we proposed GreenSlot, a parallel job scheduler for datacenters partially powered by solar energy. We implemented two versions of it: one for the SLURM scheduler and the other for the MapReduce scheduler of Hadoop. Our results demonstrated that GreenSlot's schedules consume significantly more green

energy and incur substantially lower brown energy costs than those of a conventional or even an energy-aware scheduler. With GreenSlot, the capital cost of our datacenter's solar array can be amortized in 10-11 years, whereas it would take 18-22 years to amortize those costs under the conventional or even energy-aware schedulers. Our results also showed that GreenSlot is robust to different underlying schedulers, implementations, and workloads. We conclude that green datacenters and green energy-aware scheduling can have a significant role in building a more sustainable IT ecosystem.

Although we did not consider batteries in this work, GreenSlot could be extended to leverage them and reduce brown energy consumption further. Specifically, we could extend it to run jobs at low cost (corresponding to battery losses) during slots when green energy is not being produced but the batteries are sufficiently charged.

References

[1] J. Koomey, Growth in Data Center Electricity Use 2005 to 2010, analytic Press (2011).

[2] J. Mankoff, R. Kravets, E. Blevis, Some Computer Science Issues in Creating a Sustainable World, Computer 41 (8).

[3] EcobusinessLinks, Green Web Hosts, `http://www.ecobusinesslinks.com/green_webhosts/` (Retrieved in 2013).

[4] Apple, Apple and the Environment, `http://www.apple.com/environment/renewable-energy/` (2013).

[5] Data Center Knowledge, Data Centers Scale Up Their Solar Power, `http://www.datacenterknowledge.com/archives/2012/05/14/data-centers-scale-up-their-solarpower/` (2012).

[6] US Department of Energy, 2010 Solar Technologies Market Report, Tech. rep. (2011).

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

[8] UK Government, Carbon Reduction Commitment, `http://www.carbonreductioncommitment.info/`.

[9] I. Goiri, W. Katsak, K. Le, T. D. Nguyen, R. Bianchini, Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy, in: ASPLOS, 2013.

[10] A. Jossen, J. Garche, D. Sauer, Operation Conditions of Batteries in PV Applications, Solar Energy 76 (6).

[11] R. Davis, A. Burns, A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems, Tech. Rep. YCS-2009-443, Dept. of Comp. Science, University of York (2009).

[12] A. Yoo, M. Jette, M. Grondona, SLURM: Simple Linux Utility for Resource Management, in: JSSPP, 2003.

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

[14] M. Arlitt, C. Bash, Y. Blagodurov, S. Chen, T. Christian, D. Gmach, C. Hyser, N. Kumari, Z. Liu, M. Marwah, A. McReynolds, C. Patel, A. Shah, Z. Wang, R. Zhou, Towards the Design and Operation of Net-Zero Energy Data Centers, in: ITherm, 2012.

[15] B. Aksanli, J. Venkatesh, L. Zhang, T. Rosing, Utilizing Green Energy Prediction to Schedule Mixed Batch and Service Jobs in Data Centers, in: HotPower, 2011.

[16] I. Goiri, K. Le, T. Nguyen, J. Guitart, J. Torres, R. Bianchini, GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks, in: Eurosys, 2012.

[17] K. Kant, M. Murugan, D. H. C. Du, Willow: A Control System for Energy and Thermal Adaptive Computing, in: IPDPS, 2011.

[18] A. Krioukov, S. Alspaugh, P. Mohan, S. Dawson-Haggerty, D. Culler, R. Katz, Design and Evaluation of an Energy Agile Computing Cluster, Tech. Rep. EECS-2012-13, University of California at Berkeley (January 2012).

[19] C. Li, A. Qouneh, T. Li, iSwitch: Coordinating and Optimizing Renewable Energy Powered Server Clusters, in: ISCA, 2012.

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

[21] A. Krioukov, C. Goebel, S. Alspaugh, Y. Chen, D. Culler, R. Katz, Integrating Renewable Energy Using Data Analytics Systems: Challenges and Opportunities, Bulletin of the IEEE Computer Society Technical Committee.

[22] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, C. Hyser, Renewable and Cooling Aware Workload Management for Sustainable Data Centers, in: SIGMETRICS, 2012.

[23] K. Le, R. Bianchini, M. Martonosi, T. D. Nguyen, Cost- And Energy-Aware Load Distribution Across Data Centers, in: HotPower, 2009.

[24] K. Le, O. Bilgir, R. Bianchini, M. Martonosi, T. D. Nguyen, Capping the Brown Energy Consumption of Internet Services at Low Cost, in: IGCC, 2010.

[25] K. Le, J. Zhang, J. Meng, Y. Jaluria, T. D. Nguyen, R. Bianchini, Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds, in: SC, 2011.

[26] Z. Liu, M. Lin, A. Wierman, S. Low, L. Andrew, Greening Geographical Load Balancing, in: SIGMETRICS, 2011.

[27] Y. Zhang, Y. Wang, X. Wang, GreenWare: Greening Cloud-Scale Data Centers to Maximize the Use of Renewable Energy, in: Middleware, 2011.

[28] N. Deng, C. Stewart, D. Gmach, M. Arlitt, J. Kelley, Adaptive Green Hosting, in: ICAC, 2012.

[29] C. Ren, D. Wang, B. Urgaonkar, A. Sivasubramaniam, Carbon-Aware Energy Capacity Planning for Datacenters, in: MASCOTS, 2012.

[30] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, B. Maggs, Cutting the Electric Bill for Internet-Scale Systems, in: SIGCOMM, 2009.

[31] D. Feitelson, L. Rudolph, U. Schwiegelshohn, Parallel Job Scheduling – A Status Report, in: JSSPP, 2004.

[32] D. Talby, D. Feitelson, Supporting Priorities and Improving Utilization of the IBM SP2 Scheduler Using Slack-Based Backfilling, in: IPDPS, 1999.

[33] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn,

A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, Mapping Abstract Complex Workflows Onto Grid Environments, Journal of Grid Computing 1 (1).

[34] M. Islam, Qos in parallel job scheduling, Ph.D. thesis, Dept. of Computer Science and Engineering, Ohio State University (2008).

[35] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, R. Buyya, Libra: A Computational Economy-Based Job Scheduling System for Clusters, Software Practice and Experience 34 (6).

[36] C. Lee, Y. Schwartzman, J. Hardy, A. Snavely, Are User Runtime Estimates Inherently Inaccurate?, in: JSSPP, 2004.

[37] A. W. Mu'alem, D. G. Feitelson, Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling, IEEE Transactions on Parallel and Distributed Systems 12 (6).

[38] D. Tsafrir, Y. Etsion, D. G. Feitelson, Backfilling Using System-Generated Predictions Rather Than User Runtime Estimates, IEEE Transactions on Parallel and Distributed Systems 18 (6).

[39] J. Leverich, C. Kozyrakis, On the Energy (In)efficiency of Hadoop Clusters, in: HotPower, 2009.

[40] R. T. Kaushik, M. Bhandarkar, K. Nahrstedt, Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System, in: CloudCom, 2010.

[41] W. Lang, J. M. Patel, Energy Management for MapReduce Clusters, in: VLDB, 2010.

[42] S. Jebaraj, S. Iniyan, A Review of Energy Models, Renewable and Sustainable Energy Reviews 10 (4).

[43] N. Sharma, J. Gummeson, D. Irwin, P. Shenoy, Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems, in: SECON, 2010.

[44] D. Lifka, The ANL/IBM SP Scheduling System, in: JSSPP, 1995.

[45] O. Flores, M. Orozco, NucleR: A Package for Non-Parametric Nucleosome Positioning, Bioinformatics 27 (15).

[46] P. Ranganathan, P. Leech, D. Irwin, J. Chase, Ensemble-level Power Management for Dense Blade Servers, in: ISCA, 2006.

[47] I. Rodero, F. Guim, J. Corbalan, Evaluation of Coordinated Grid Scheduling Strategies, in: HPCC, 2009.

[48] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay Scheduling: a Simple Technique for Achieving Locality and Fairness in Cluster Scheduling, in: Eurosys, 2010.

[49] O. OMalley, A. Murthy, Winning a 60 Second Dash with a Yellow Elephant, in: Sort Benchmark, 2009.

[50] G. Wang, A. R. Butt, H. Monti, K. Gupta, Towards Synthesizing Realistic Workload Traces for Studying the Hadoop Ecosystem, in: MASCOTS, 2011.