GreenSlot: Scheduling Energy Consumption in Green Datacenters

Íñigo Goiri[†‡], Kien Le[‡], Md. E. Haque[‡], Ryan Beauchea[‡],
Thu D. Nguyen[‡], Jordi Guitart[†], Jordi Torres[†], and Ricardo Bianchini[‡]

[†]Universitat Politècnica de Catalunya
Barcelona Supercomputing Center
{jguitart,torres}@ac.upc.edu

[‡]Department of Computer Science
Rutgers University
{goiri,lekien,mdhaque,rybeauch,tdnguyen,ricardob}@cs.rutgers.edu

Technical Report DCS–TR–686, Dept. of Computer Science, Rutgers University, May 2011

ABSTRACT

In this paper, we propose GreenSlot, a parallel batch job scheduler for 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 brown energy must be used to avoid deadline violations, the scheduler selects times when brown electricity is cheap. Our results for production scientific workloads demonstrate that GreenSlot can increase green energy consumption by up to 117% and decrease energy cost by up to 39%, compared to EASY backfilling. 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

As of 2006, the datacenters in the United States were consuming 61.4 Billion kWh per year, an amount of energy that is equivalent to that consumed by the entire transportation manufacturing industry (the industry that makes airplanes, ships, cars, trucks, and other means of transportation) [1]. Even worse, datacenter energy consumption has increased significantly since 2006.

Large Internet companies (e.g., Google and Microsoft) have made significant strides in improving 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 datacenters [1], 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.

Unfortunately, the cost of the energy consumed by these datacenters is not the only problem. Their energy consumption also contributes to climate change, since most of the electricity produced in the US and around the world comes from burning coal and natural gas, which are greenhouse-gas-intensive approaches to energy production [2]. In fact, a single server can have roughly the same carbon footprint as a Sports Utility Vehicle [3].

These cost and environmental concerns have been prompting many "green" energy initiatives. One initiative is for datacenters to either generate their own solar/wind energy or draw power directly from a nearby green solar/wind farm. This approach is becoming popular, as demonstrated by the many small and medium datacenters (partially or completely) powered by solar and/or wind energy that are popping up all over the globe (see *http://www.ecobusiness-links.com/green_web_hosting.htm* for a partial list). We expect that this trend will continue, as these technologies' capital costs keep decreasing (e.g., the cost of solar energy has decreased by 7-fold in the last two decades [4]) and governments continue to provide generous incentives for green power generation (e.g., federal and state incentives in New Jersey can reduce capital costs by 60% [5]). 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 [6].

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 [7]; (3) batteries use 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. Based on these observations, it is clear that 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 a datacenter's computational workload to match the green energy supply. In particular, we design a parallel batch job scheduler, called GreenSlot, for 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, GreenSlot schedules jobs for times when

brown electricity is cheap. In more detail, GreenSlot combines solar energy prediction, energy-cost-awareness, and least slack time first (LSTF) job ordering [8]. 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. As it should be clear by now, GreenSlot departs significantly from most job schedulers, which seek to minimize completion times or bounded slowdown.

We implement GreenSlot as an extension of the widely used SLURM scheduler for Linux [9]. Our experiments use real scientific workloads (and their actual arrival times and deadlines) that are in current use at the Life Sciences Department of the Barcelona Supercomputing Center. We model the datacenter's solar array as a properly scaled-down version of the Rutgers solar farm in New Jersey. The brown electricity prices are from a power company in New Jersey. Our results demonstrate that GreenSlot accurately predicts the amount of solar energy to become available. The results also demonstrate that GreenSlot can increase green energy consumption and decrease energy cost by up to 117% and 39%, respectively, compared to a standard backfilling scheduler.

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:

- Introduce GreenSlot, a batch job scheduler for datacenters partly powered by solar energy;
- Introduce job scheduling that is aware of green energy;
- Introduce job scheduling that is aware of brown electricity prices; and
- Present extensive results isolating the impact of different aspects of the scheduler.

The remainder of the paper is organized as follows. The next section provides background on solar energy generation, its use to power computer systems, and discusses its costs. Section 3 discusses the related work and compares GreenSlot to other job schedulers. Section 4 presents our approach to solar energy prediction, our scheduling policy, and GreenSlot. Section 5 describes our experimental methodology and results. Finally, Section 6 draws our conclusions and mentions a few avenues for future work.

2. BACKGROUND

Solar energy and its use in datacenters. Solar energy is perhaps the most promising of the clean energy technologies, 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 as solar energy at our 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 most 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 extra 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 a general setup with all the components present. The solar panels can be connected to batteries for storing excess energy during periods of sunlight and discharging it during other periods. Since current server and cooling equipment run on alternating current (AC), the DC electricity produced by the panels must be converted to AC electricity using an inverter. When the datacenter must be operational even when solar energy (stored in batteries or otherwise) is not available, it must also be connected to the electrical grid via a grid-tie device. Where net metering is available, it is possible to feed excess solar energy into the grid for a reduction in brown energy costs.

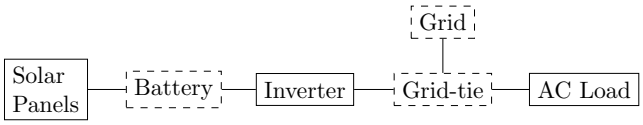

Figure 1: Components of a solar-powered computer system. Dashed boxes represent optional components.

The design we study in this paper does not include batteries or net metering, for the reasons we mentioned in the Introduction. In this scenario, any 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 governmental incentives in NJ, the current capital cost of installing solar panels for the datacenter we model can be amortized by savings in brown energy cost in 11 years of operation. This amortization period is substantially shorter than the typical lifetime of the solar panels, 20-30 years. The period will be even shorter in the future, as solar costs continue to decrease at a rapid pace [4].

Brown electricity prices. Datacenters often contract with their power companies to pay variable brown electricity 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 electricity 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. Some previous works have addressed these issues [10, 11, 12, 13]. In a short position paper, Stewart and Shen discuss how to maximize green energy use in datacenters [13]. However, their main focus was on request distribution in multi-datacenter interactive Internet services. Similarly, [10, 11, 12] focused solely on these multi-datacenter services. Our work differs from these previous efforts in many ways. Specifically, only [12] considered green energy predictions, and only [10, 11, 12] considered variable brown electricity prices. None of these papers considered batch job scheduling. Batch jobs typically run longer than interactive service requests and often have loose deadlines, thereby increasing the opportunity to exploit green energy.

Another key difference between our paper and these previous efforts is that we focus on a single datacenter. Also targeting a single datacenter, Blink [14] 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. In addition, our approach for managing green energy consumption is through job scheduling, rather than server power state scheduling.

In contrast with these higher level approaches, SolarCore [15]

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 electricity prices. Most of the works that have considered variable electricity prices have targeted request distribution across multiple datacenters in interactive Internet services [10, 11, 12, 16]. The exception is our recent work [17], which considers variable electricity prices in multi-datacenter high-performance computing clouds. Our current work differs from these previous efforts as it seeks to maximize green energy use, predict green energy availability, and schedule batch jobs within a single datacenter.

GreenSlot vs. other job schedulers. GreenSlot has a few unique characteristics, compared to other job schedulers, e.g. [9, 18]. 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 [19] 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 electricity prices.*

Second, GreenSlot borrows ideas from (soft) real-time systems: (1) jobs and/or workflows (i.e., sequences of related jobs [20]) have deadlines by which they shall complete; (2) it keeps the queued jobs in LSTF order [8]; 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., [21, 22]),* most of them typically do not.

Many scientific workloads have natural deadlines, even if they are not always explicitly stated. For example, it is common to submit jobs before leaving work on Friday and not needing their results until Monday. Importantly, *deadlines provide the flexibility that allows GreenSlot to achieve its goals.* Nevertheless, users that need their jobs/workflows to complete quickly can also use GreenSlot. They simply need to specify tight deadlines. However, the presence of these jobs/workflows reduces flexibility, and may cause brown energy consumption and/or energy cost to increase.

GreenSlot suspends jobs that have outrun their allowed run times, instead of canceling them like most other schedulers do. It does so because these jobs have already consumed energy, so it would be wasteful to cancel them. To encourage users not to overestimate the expected run times, we envision a pricing model in which users 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 in case GreenSlot is unable to meet a deadline, the datacenter operator would reimburse the user for an amount proportional to the length of the violation.

4. SCHEDULING IN GREEN DATACENTERS

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

Figure 2 illustrates the behavior of GreenSlot (bottom), in comparison to a standard EASY backfilling scheduler (top) for three jobs. Each rectangle represents the number of nodes and time that each job will likely require. The 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 the execution of some jobs (as long as they do not violate 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 electricity (not shown). GreenSlot is beneficial because datacenters are not fully utilized at all times.

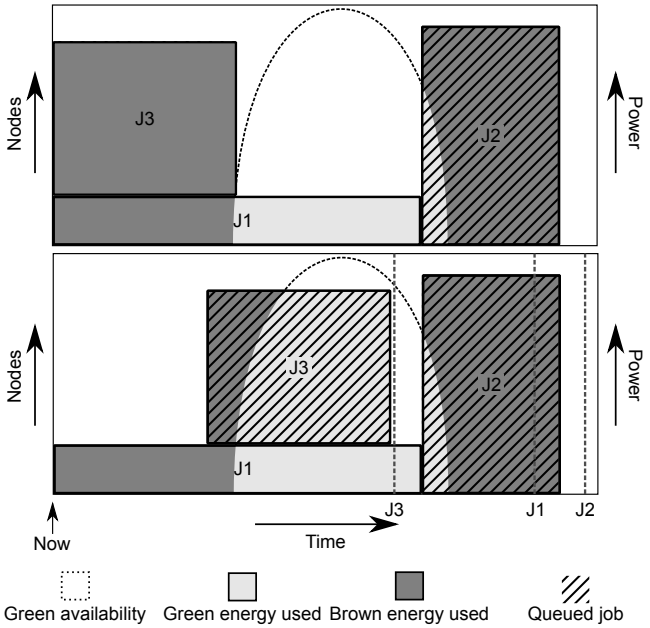

Figure 2: Scheduling 3 jobs (J1-J3) with backfilling (top) and GreenSlot (bottom). The jobs' deadlines are the vertical lines.

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

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 electricity prices often depend on time of use, as aforementioned. When the price of brown electricity 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.

GreenSlot is greedy in two ways: (1) it schedules jobs that are

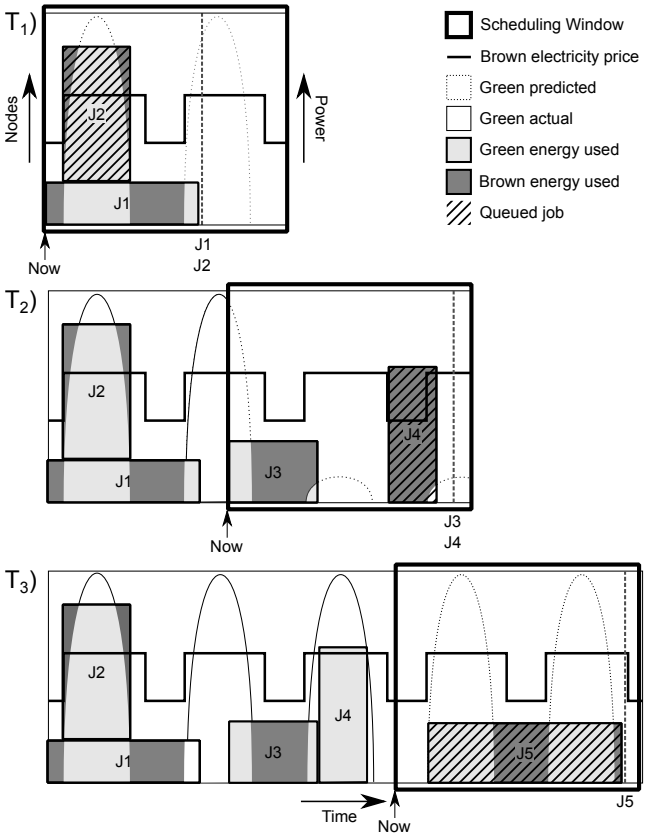

Figure 3: GreenSlot scheduling window at times T_1 (top), T_2 (middle), and T_3 (bottom).

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 on the cluster, 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 very 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 solar energy 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 solar energy, J4 is scheduled for the following day during a period of cheap brown electricity. More than a day later than T_2, at T_3, we see that GreenSlot initially mispredicted the amount of solar energy at time T_2. It later adjusted its prediction and ran J4 earlier. Finally, we also see J5 queued waiting for solar energy to become available.

Details. Figure 4 presents the GreenSlot scheduling algorithm. Line 0 represents 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. Although this is not shown in Figure 4, jobs that take longer than this extended amount of time are suspended and must be re-started by hand. Our goal is to tolerate some inaccuracy in the information provided by users, 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 that has three phases that must be executed without overlap, each of which is expected to take 60 minutes. Suppose also 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. GreenSlot first determines whether its predictions for the amount of solar energy were accurate in the most recent slot (line 2). If the predictions were inaccurate, it 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). In addition, a new schedule is 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 datacenter administrator determines the average energy consumed by the jobs of each workflow based on their previous executions. We plan to automate this energy 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 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 electricity prices (line 16).

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 electricity 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.

```
0.  Users specify the 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 the 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 this 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 this window should only be scheduled in this 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).

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 scheduling 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 very 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. A potential drawback of GreenSlot is that it may reject more jobs or miss more deadlines than a scheduler that delays fewer jobs. However, as our sensitivity study in Section 5.2 demonstrates, this is only likely to occur in datacenters with uncommonly 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:

$$E_p(t) = f(w(t))B(t) \quad (1)$$

where $E_p(t)$ is the amount of solar 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 solar energy expected under ideal sunny conditions.

We implement solar energy prediction using the above model at the granularity of an hour. We use weather forecasts widely available from sites such as The Weather Channel weather.com and Weather Underground wunderground.com to instantiate $w(t)$. These sites provide hourly predictions for up to 48 hours in the future. Each prediction includes a string describing the forecasted condition such as "sunny", "cloudy", or "scatter thunderstorm". This string is the output of $w(t)$.

We use historical data to instantiate both $B(t)$ and $f(t)$. Specifically, for a given hour t, we use the actual weather conditions and energy generated during the month centered on t from the previ-

ous year. We choose this period around t, called $H(t)$, to account for seasonal effects. We set $B(t)$ to the maximum observed energy generated for the same hour of any day in $H(t)$. For each different weather condition wc, we compute $f(wc)$ using recorded data from all hours in $H(t)$ with the same weather condition.

Unfortunately, weather forecasts can be wrong. For example, we have observed that predictions for thunderstorms are frequently inaccurate and can remain inaccurate throughout a day; i.e., the forecast continues to predict thunderstorm hour-by-hour but the storm does not arrive during that day. Further, weather is not the only factor that affect 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 $f(t)$. Specifically, we assume that the recent past can predict the near future, and compute $f(t)$ 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 $f(t)$ for the remainder of the current day. Beyond the current day, we always instantiate $f(t)$ 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 [23]. However, these models tend to be inaccurate at medium time scales [24]. Based on this observation, Sharma *et al.* proposed a simple model based on historical data and weather forecasts [24]. Our approach is similar, but also embodies error correction based on the recent green energy production.

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 8GBytes of memory, 1 7200rpm SATA disk, and a 1Gbit Ethernet card. GreenSlot runs on an additional server. The servers are connected by a Gigabit Ethernet switch.

GreenSlot extends the popular SLURM scheduler for Linux with roughly 2800 uncommented lines of Python code. We study 2 versions of GreenSlot: "GreenOnly", which makes decisions based on green energy availability, but not variable brown electricity prices; and "GreenVarPrices", which considers both green energy and variable brown electricity prices.

For comparison, we also study a variant of EASY backfilling [25] 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. Like GreenSlot, backfilling assigns a 20% tolerance to the user-estimated job run times. If a job's run time estimate and tolerance are exceeded, backfilling cancels the job.

Workloads. Our 3 workloads are currently in production use at the Life Sciences Department at the Barcelona Supercomputing Center [26]. 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 1.2GBytes, the intermediate file sizes are 800MBytes each, and the final output size is 100MBytes.

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 throughout the paper.

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 workloads. Nevertheless, in Section 5.2, we do study workload variations that include multi-node jobs to demonstrate GreenSlot's behavior in the face of such jobs. In the same section, we also consider workload arrivals that are staggered over the week, rather than clustered on Monday. Finally, most of our experiments assume that the user-provided estimates of job run time are exactly the run times listed above. Nevertheless, in that section, we also study the impact of inaccuracies in these estimates. In those experiments, we create the inaccuracies by changing the estimates, not the jobs' actual run times.

Power consumption and solar panel array. Using an accurate Yokogawa multimeter, 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. In addition, the switch consumes 55W and the low-power server that runs GreenSlot consumes roughly 30W. Overall, the common-case peak power consumption of our system for our workloads is 1765W = 16 $\times$ 105W + 55W + 30W. When some servers are idle, GreenSlot sends each of them to S3 state, which consumes 8.6W. Transitioning into and out of S3 takes 7 seconds.

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 de-rating) 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 production down to 10 solar panels capable of producing 2.3kW of power. We selected this scaled size because, after de-rating, 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

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.

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), the best week for our system as compared to backfilling (starting on August 23rd 2010), and the worst week for our system compared to backfilling (starting on March 7th 2010). We call these weeks "Most", "Average", "Best", and "Worst", respectively.

Brown electricity prices. We assume the most common type of variable brown electricity pricing, namely on-peak/off-peak pricing. In on-peak/off-peak pricing, electricity 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 Best weeks.

Accelerating and validating the experiments. It would be impossible for us to perform all of the experiments in this paper in real time. This would require 200 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.

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 GreenVarPrices scheduling our real workloads with their estimated run times and deadlines. The corresponding accelerated run uses this same version of GreenSlot, but 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. 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 Results

This section presents our experimental results. First, we evaluate the accuracy of our solar energy predictions independently of GreenSlot. Second, we isolate the impact of being aware of green energy by comparing GreenOnly with backfilling. These results also assess the impact of the quality of green energy predictions on our scheduling. Third, we study GreenVarPrices to isolate the benefit of being aware of brown electricity prices. Fourth, we study the impact of the workload characteristics on the GreenVarPrices results. Fifth, we study the impact of the datacenter utilization on GreenVarPrices. Finally, we quantify the impact of poor run time estimates by users.

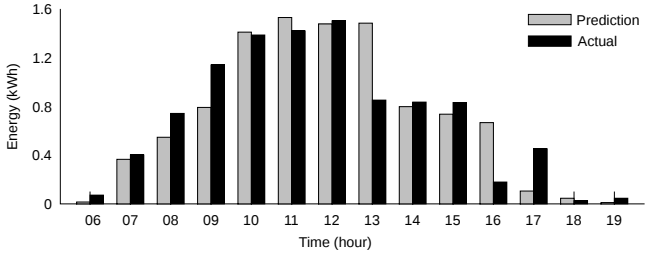

Figure 5: Predicted and actual energy for June 1, 2010.

Throughout these experiments, *backfilling and GreenSlot do not violate any deadlines,* except when we explore high datacenter utilizations to purposely cause violations.

Predicting solar energy. 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 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 ($\sim$14.8kWh). Further, while prediction accuracy degrades with prediction horizon, this degradation is quite small beyond 3 hours. Even when predicting 48 hours ahead, the median daily error is 19.0%.

Of the 4 weeks we use to study GreenSlot in detail, week Best has the best prediction accuracy, with a median 1-hour ahead prediction error of 9.3%, while week Worst 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.

Figure 5 plots the 1-hour ahead predictions and actual energy generation for an interesting day to demonstrate the workings of our predictor. Throughout the morning of the day, the weather forecast predicted "mostly cloudy" conditions even though the actual cloud cover was light. This led to significant prediction errors at hours 8 and 9. Our predictor detected this weather misprediction and switched to its alternative prediction mode, which improved predictions for hours 10-12. When a light rain developed during hour 13, actual production dropped, leading to a large prediction error (production during hour 12 was not a good predictor for hour 13). Our predictor adjusted appropriately for hours 14 and 15. A thunderstorm developed during hour 16, again leading to a drop in production and a prediction error. The thunderstorm stopped in hour 17, leading to a misprediction in the reverse direction. This day illustrates the fact that GreenSlot's use of two predictors enables it to quickly correct prediction errors.

Scheduling for solar energy and impact of predictions. Figure 6 shows the behavior of the backfilling scheduler for our real 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 electricity prices (right). The figure depicts the green and brown energy consumptions using ar-

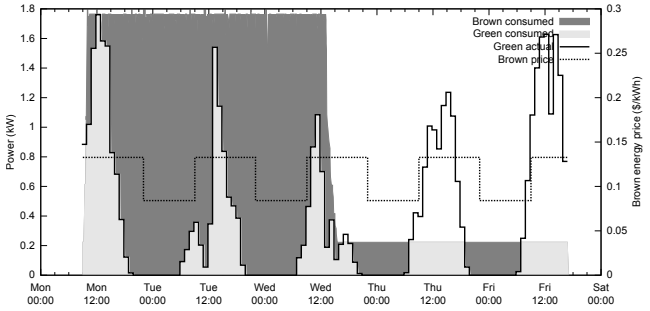

Figure 6: Backfilling scheduler and Average week.

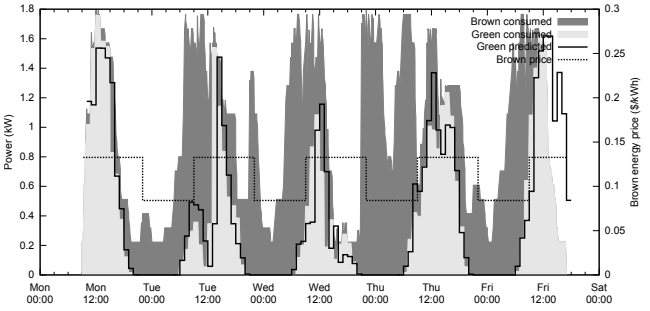

Figure 7: GreenOnly scheduler and Average week.

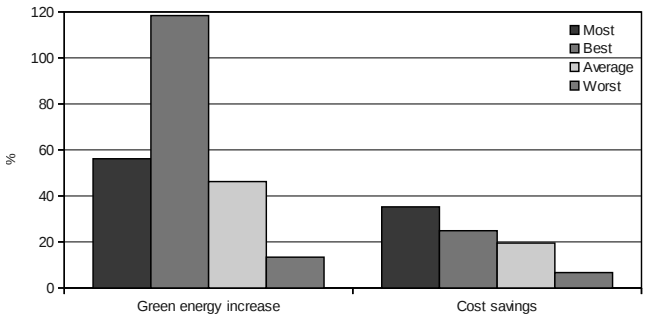

Figure 8: GreenOnly's green energy increase and cost savings.

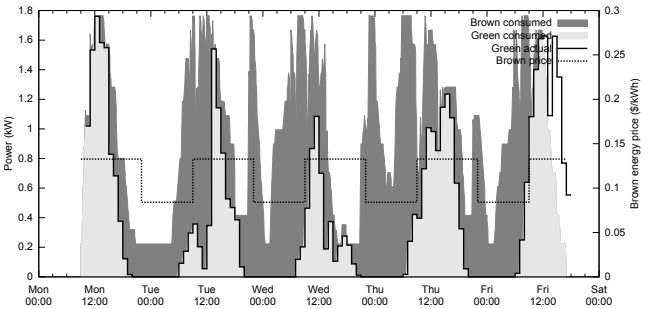

Figure 9: GreenOnly with actual green energy availability.

eas colored light gray and dark gray, respectively. The two line curves represent the green energy available (labeled "Green actual") and the brown electricity price ("Brown price").

The figure shows that the cluster utilization is roughly 50%, which is comparable to (or even somewhat higher than) many real scientific-computing datacenters and grids [27, 28]. As backfilling schedules the workloads to complete as soon as possible, it heavily uses the machines 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" or "base" energy [29]. However, backfilling 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 7 depicts the behavior of GreenOnly, under the same conditions as in Figure 6. Note that, 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 sometimes do not match the actual green energy available.

A comparison between Figures 6 and 7 clearly illustrates how GreenOnly is capable of using substantially more green energy than backfilling, 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 backfilling in this experiment. Although GreenOnly does not explicitly consider brown electricity prices in making decisions, its energy cost savings reach 20% compared to backfilling. More than 80% of these cost savings comes from replacing brown energy with green energy.

The results for the other weeks are similar, as we can see in Figure 8. The figure shows two sets of 4 bars. Each bar in a set represents a week. The set on the left represents the increase in green energy consumption, whereas the set on the right represents the energy cost savings. Overall, GreenOnly increases green energy consumption between 13% and 118%, whereas it reduces costs between 7% and 35%. Interestingly, GreenOnly improves on backfilling even for the worst-case week (Worst) 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 7 well. To quantify the impact of prediction accuracy, consider Figure 9. 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 7 and 9 shows similar schedules. Overall, we find that perfect knowledge increases green energy use and decreases cost both by 1%. These results suggest that GreenSlot can be improved by more accurate (and perhaps less practical) green energy predictions, but only slightly. Thus, this experiment is the only one in which we consider perfect knowledge of green energy availability.

Scheduling for solar energy and brown electricity prices. So far, we have studied scheduling that does not explicitly exploit variable brown electricity prices. However, GreenSlot can reduce costs further when brown electricity 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 10 shows the behavior of GreenVarPrices again for our real workloads, the Average week, and accurate job run time estimates. Comparing this figure against Figure 7, one can clearly see that GreenVarPrices moves many jobs that must consume brown energy to periods with cheap brown electricity. For example, GreenOnly runs many jobs on Tuesday night, Wednesday night, and Thursday night that consume expensive brown electricity. Those jobs get scheduled during periods of cheap electricity under GreenVarPrices. As a result, GreenVarPrices exhibits higher energy cost savings of 25% compared to backfilling for this week, while consuming almost exactly the same amount of green energy as GreenOnly.

GreenVarPrices achieves positive results for the other weeks as well, as illustrated in Figure 11. Overall, the GreenVarPrices cost savings range between 13% and 39%, whereas its increases in green

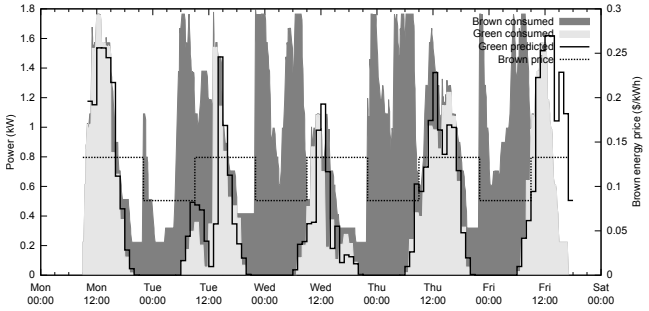

Figure 10: GreenVarPrices scheduler and Average week.

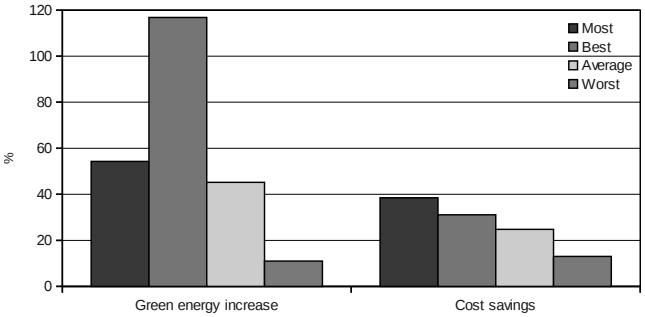

Figure 11: GreenVarPrices' green energy increase and cost savings.

energy consumption range between 11% and 117%.

A comparison between Figures 8 and 11 illustrates the benefit of considering brown electricity prices explicitly in GreenSlot. As one would expect, doing so decreases costs further with respect to GreenOnly. To isolate GreenSlot's ability to exploit cheap brown electricity in the absence of green energy, we also consider an idealized week with absolutely no solar energy. For this week, GreenVarPrices reduces energy cost by 13% with respect to backfilling.

Impact of workload characteristics. The results so far have assumed workloads with the exact characteristics they have in the field. However, one may argue that these characteristics are favorable to GreenSlot, in that there are no multi-node jobs and all workflows are submitted on Monday. Here, we create variations of the workloads to assess the benefits of GreenVarPrices under other scenarios. In particular, we create a variation called "Staggered", in which the workflow submissions are staggered across the week. In the variation called "Multi-node", we assume that the 8 compute jobs of Workload3 should be gang-scheduled together and take as long as the longest compute job.

Figures 12 and 13 present the behavior of backfilling and GreenVarPrices, respectively, for Staggered, the Average week, and accurate run time estimates. With Staggered, backfilling lucks into exploiting more green energy. However, most jobs end up scheduled at periods with high brown electricity prices. In contrast, GreenVarPrices smartly spreads the load to take significant advantage of both green energy and cheaper brown electricity. Overall, the GreenVarPrices green energy increase and energy cost savings are 19% and 30%, respectively.

Backfilling behaves similarly for Multi-node and our real workloads, as depicted in Figure 14, again for the Average week and accurate run time estimates. In contrast, Figure 15 shows that GreenVarPrices spreads the load across the week, achieving a green energy increase and a cost savings of 45% and 25%, respectively.

These results show that GreenSlot is robust to different workload characteristics, providing green energy increases of at least 19% and energy cost savings of at least 25% compared to backfilling.

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 electricity, may be forced to violate deadlines, and/or even cancel newly submitted jobs.

To investigate these effects, we perform experiments with backfilling and GreenVarPrices for four additional datacenter utilizations: 67%, 72%, 87%, and 92%. Starting with our real workloads, 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 scientific environments [27, 28].

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 backfilling at the same utilization. In contrast, backfilling 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.

Impact of inaccurate run time estimates. Our final set of experiments investigates the impact of inaccuracies in the user-provided run time estimates on backfilling and GreenVarPrices. In particular, we consider four scenarios of normally distributed inaccuracies for GreenVarPrices, our real workloads, and the Average week: (1) inaccuracies ranging between -20% and +20% with an average of 0%; (2) ranging between 0% and +20% with an average of +10%; (3) ranging between -20% and 0% with an average of -10%; and (4) ranging between -40% and 0% with an average of -20%. Recall that GreenSlot (backfilling) builds into its schedule a tolerance of +20% for every job, and suspends (cancels) any job with an inaccuracy > +20%. This is why we do not include jobs with inaccuracies > +20% in this sensitivity analysis. In turn, this means that no job in our sensitivity experiments runs longer than backfilling or GreenSlot expects.

These results show that backfilling is essentially insensitive to these inaccuracies, as the actual job run times are the same as before (only the user estimates are inaccurate). Backfilling is always capable of fully loading the system for half of the week; whenever a job finishes another can be scheduled on the same resources.

The inaccuracies have only a minor impact on GreenSlot. Compared to having accurate estimates, the largest impact occurs in scenario (1), where the energy cost increases by less than 2% and the green energy consumption stays almost exactly the same. To understand why the green energy consumption is insensitive to these inaccuracies, consider a job that should be consuming green energy at the time GreenSlot expects it to terminate (its start time plus the user-estimated run time plus the 20% tolerance). If this job completes earlier than this expected time, GreenSlot will likely schedule one or more jobs to consume the leftover green energy.

To understand why the energy cost changes only marginally, note that GreenSlot may not always change its schedule as a result of an inaccuracy; it only does so only if a change would reduce cost. On the other hand, some inaccuracies may increase cost when jobs start consuming expensive brown energy as a result. Our experiments show that these effects typically balance out for our real workloads.

6. CONCLUSIONS

In this paper, we proposed GreenSlot, a parallel job scheduler for datacenters partially powered by solar energy. GreenSlot pre-

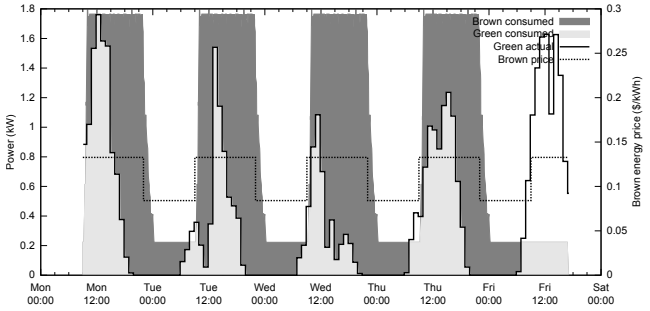

Figure 12: Backfilling for staggered workloads.

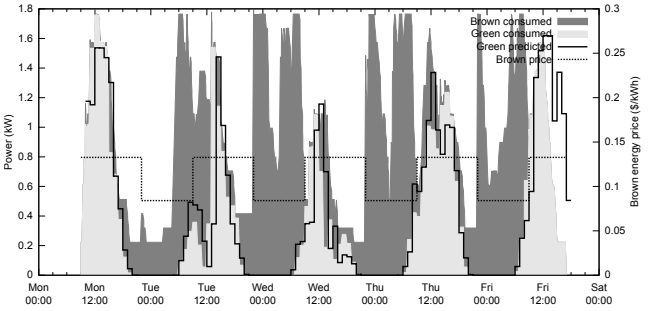

Figure 13: GreenVarPrices for staggered workloads.

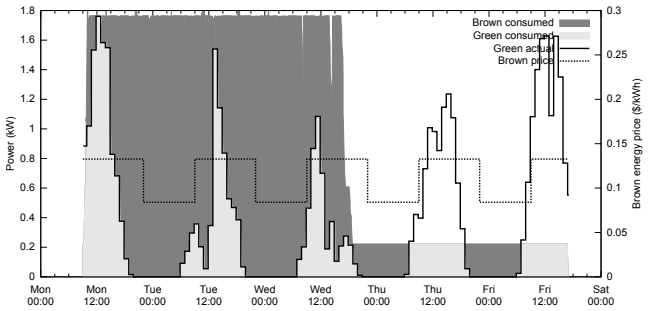

Figure 14: Backfilling for multi-node workloads.

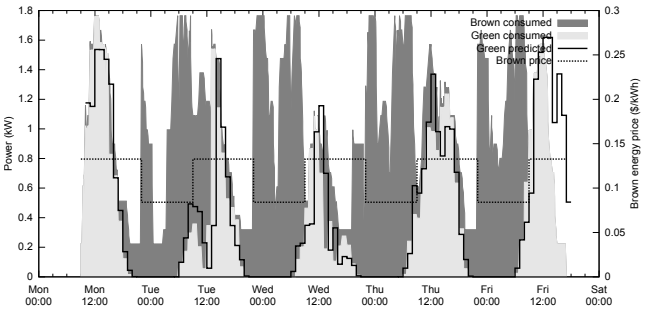

Figure 15: GreenVarPrices for multi-node workloads.

dicts solar energy availability two days into the future. Using these predictions, it schedules jobs to maximize the use of green energy and limit brown energy costs, while avoiding deadline violations. Our results demonstrated that GreenSlot's schedules consume significantly more green energy and incur substantially lower brown energy costs than those of a backfilling scheduler. With GreenSlot, the capital cost of our datacenter's solar array can be amortized in 11 years, whereas it would take 18 years to amortize those costs under backfilling. Based on our positive results, we conclude that green datacenters and green-energy-aware scheduling can have a significant role in building a more sustainable IT ecosystem.

To demonstrate this in practice, we are building a prototype micro-datacenter (80 Atom-based servers placed in a small container) powered by a 5kW solar array and the electrical grid. The micro-datacenter will use free cooling almost year-round and will be placed on the roof of our building at Rutgers. We expect to include some results from the prototype in the final version of the paper.

Our future work will further explore our approach for green energy prediction, improve GreenSlot's ability to gracefully handle very high datacenter utilizations, and extend GreenSlot for managing peak brown power consumption for those datacenters that are subject to peak brown power charges.

Acknowledgements

We would like to thank Oscar Flores and Dr. Modesto Orozco for the genomics workloads we used in this paper. We would also like to thank Dr. Julita Corbalan for comments on the paper. Finally, we are grateful to our sponsors: Spain's Ministry of Science and Technology and the European Union under contract TIN2007-60625 and grant AP2008-0264, the Generalitat de Catalunya grants 2009-SGR-980 2010-PIV-155, NSF grant CSR-0916518, and the Rutgers Green Computing Initiative.

7. REFERENCES

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

[2] Power Scorecard, "Electricity from Coal," http://www.powerscorecard.org/tech_detail.cfm?resource_id=2.

[3] Global Action Plan, "An Inefficient Truth," December 2007, http://globalactionplanorguk.site.securepod.com/upload/resource/-Exec-Summary.pdf.

[4] SolarBuzz, "Marketbuzz," 2011, http://www.solarbuzz.com/our-research/reports/marketbuzz.

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

[6] UK Government, "Carbon Reduction Commitment," http://www.carbonreductioncommitment.info/.

[7] A. Jossen, J. Garche, and D. Sauer, "Operation conditions of batteries in pv applications," *Solar Energy*, vol. 76, no. 6, pp. 759–769, 2004.

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

[9] A. Yoo and M. Jette and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing*, June 2003.

[10] K. Le, R. Bianchini, M. Martonosi, and T. D. Nguyen, "Cost-And Energy-Aware Load Distribution Across Data Centers," in *Proceedings of HotPower*, 2009.

[11] K. Le, O. Bilgir, R. Bianchini, M. Martonosi, and T. D. Nguyen, "Capping the Brown Energy Consumption of Internet Services at Low Cost," in *Proceedings of the International Green Computing Conference*, August 2010.

[12] Z. Liu, M. Lin, A. Wierman, S. Low, and L. Andrew, "Greening Geographical Load Balancing," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, June 2011.

[13] C. Stewart and K. Shen, "Some Joules Are More Precious

Than Others: Managing Renewable Energy in the Datacenter," in *Proceedings of the Workshop on Power Aware Computing and Systems*, October 2009.

[14] N. Sharma, S. Barker, D. Irwin, and P. Shenoy, "Blink: Managing Server Clusters on Intermittent Power," in *Proceeding of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2011.

[15] C. Li, W. Zhang, C. Cho, and T. Li, "SolarCore: Solar Energy Driven Multi-core Architecture Power Management," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, February 2011.

[16] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the Electric Bill for Internet-Scale Systems," in *Proceedings of SIGCOMM*, August 2009.

[17] K. Le, J. Zhang, J. Meng, Y. Jaluria, T. D. Nguyen, and R. Bianchini, "Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds," Dept. of Computer Science, Rutgers University, Tech. Rep. DCS–TR–680, November 2010, Revised April 2011.

[18] D. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel Job Scheduling – A Status Report," in *Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.

[19] D. Talby and D. Feitelson, "Supporting Priorities and Improving Utilization of the IBM SP2 Scheduler Using Slack-Based Backfilling," in *Proceedings of the International Parallel Processing Symposium*, April 1997.

[20] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows Onto Grid Environments," *Journal of Grid Computing*, vol. 1, no. 1, pp. 25–39, 2003.

[21] M. Islam, "Qos in parallel job scheduling," Ph.D. dissertation, Dept. of Computer Science and Engineering, Ohio State University, 2008.

[22] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, "Libra: A Computational Economy-Based Job Scheduling System for Clusters," *Software Practice and Experience*, vol. 34, no. 6, May 2004.

[23] S. Jebaraj and S. Iniyan, "A Review of Energy Models," *Renewable and Sustainable Energy Reviews*, vol. 10, no. 4, August 2006.

[24] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy, "Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems," in *Proceeding of the International Conference on Sensor Mesh and Ad Hoc Communications and Networks*, June 2010.

[25] D. Lifka, "The ANL/IBM SP Scheduling System," in *Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing*, 1995.

[26] O. Flores and M. Orozco, "NucleR: A Package for Non-Parametric Nucleosome Positioning," 2011, submitted for publication.

[27] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level Power Management for Dense Blade Servers," in *Proceedings of the International Symposium on Computer Architecture*, June 2006.

[28] I. Rodero, F. Guim, and J. Corbalan, "Evaluation of Coordinated Grid Scheduling Strategies," in *Proceedings of the International Conference on High-Performance Computing and Communications*, 2009.

[29] L. A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *IEEE Computer*, vol. 40, no. 12, December 2007.