GreenPar: Scheduling Parallel High Performance Applications in Green Datacenters

Md E. Haque[†] Íñigo Goiri[‡*] Ricardo Bianchini[†‡] Thu D. Nguyen[†]

[†]Rutgers University
{mdhaque, tdnguyen}@cs.rutgers.edu

[‡]Microsoft Research
{inigog, ricardob}@microsoft.com

ABSTRACT

We propose GreenPar, a scheduler for parallel high-performance applications in datacenters partially powered by on-site generation of renewable ("green") energy. GreenPar schedules the workload to maximize the green energy consumption and minimize the grid ("brown") energy consumption, while respecting a performance service-level agreement (SLA). When green energy is available, GreenPar increases the resource allocations of active jobs to reduce runtimes. When using brown energy, GreenPar reduces resource allocations within the constraints imposed by the performance SLA to conserve energy. GreenPar makes its decisions based on the speedup profile of each job. We have implemented GreenPar in a real solar-powered datacenter. Our results show that GreenPar can increase the green energy consumption and reduce both the average job runtime and the brown energy consumption, compared to schedulers that are oblivious to on-site green energy.

Categories and Subject Descriptors

C.4 [**Computer Systems Organizations**]: Performance of Systems; D.m [**Software**]: Miscellaneous

Keywords

Renewable energy; energy-aware scheduling; datacenters.

1. INTRODUCTION

Datacenters consume an enormous amount of electricity, and this consumption is growing rapidly. Estimates for 2010 indicate that datacenters consumed 1.5% of the total electricity used world-wide, and that this usage increased by 56% over the previous 5 years [28]. This consumption translates into high carbon emissions, since most of the electricity is produced by burning fossil fuels.

With increasing societal awareness of emissions and climate change, there is an increasing demand for cleaner products and services. As a result, there is a rising interest in powering datacenters at least partially using on-site generation of renewable ("green") energy from sources such as wind and solar. For example, Apple and McGraw-Hill have built 40MW [6] and 14MW [10] co-located solar arrays, respectively, for their datacenters. Green House Data [19], AISO [1], and GreenQloud [20] are cloud providers that operate green datacenters mostly (or entirely) powered by wind and/or solar energy.

Such "green" datacenters hold great promise for reducing the environmental impact and electricity cost of datacenter computing. For example, Goiri et al. argue that the installed capital cost for the solar energy system of a green microdatacenter can be recovered from electricity cost savings in less than 10 years [16]. However, an important research challenge arising in these green datacenters is that generation of green energy from sources such as solar and wind is variable. For example, photovoltaic (PV) solar energy is only available during the day and the amount produced depends on the weather and season. One approach for mitigating this variability is to store green energy in batteries or the grid via net metering. However, this approach has significant disadvantages: (1) batteries are expensive[1] and common types of batteries (e.g., lead-acid) are harmful for the environment, (2) batteries incur energy losses, (3) net metering incurs losses and is not available everywhere, and (4) where net metering is available, the power company may pay less than the retail electricity price for the green energy.

Given the above disadvantages, in this paper, we investigate how to manage the computational workload to better match the energy demand to the energy supply. Specifically, we propose and evaluate GreenPar, a scheduler for parallel high-performance computing (HPC) workloads in green datacenters. GreenPar seeks to maximize green energy consumption without the need for energy storage, thereby increasing the benefits of on-site generation.[2] GreenPar also seeks to minimize "brown" grid energy consumption (when there is insufficient green energy) while respecting a performance service-level agreement (SLA). GreenPar increases the resource allocations of active parallel jobs to reduce runtimes when green energy is available. When there is insuf-

*This work was done while Goiri was at Rutgers University.

ICS'15, June 8–11, 2015, Newport Beach, CA, USA.

http://dx.doi.org/10.1145/2751205.2751221.

[1]Goiri et al. found that the cost of batteries is currently not amortizable when they are used as a power source in a green datacenter, unless there is enough performance slack for deferring the workload [16].

[2]The datacenter still has batteries for handling grid failures, but GreenPar does not use them because that would require extra capacity and frequent charge/discharge may shorten their lifetimes.

ficient green energy, GreenPar reduces resource allocations within the constraints imposed by the performance SLA to conserve brown energy. GreenPar relies on the jobs' speedup profiles in making its decisions. A specific performance SLA that we consider in this paper is a maximum runtime slowdown percentage; for example, the SLA might state that a job's execution cannot be slowed down by more than 10% in order to conserve brown energy.

GreenPar can reduce brown energy consumption *even when it is not allowed to slow down the jobs*. It can do this by first allocating more resources than requested to a parallel job when green energy is available, increasing the job's rate of progress and thus letting it accumulate performance "slack." Subsequently, when running on brown energy, GreenPar can conserve energy by reducing the resource allocation of the job, using the accumulated performance slack to avoid lengthening its overall completion time.

We have designed four green-energy-aware scheduling policies for GreenPar, targeting solar-powered green datacenters. Each policy assumes a different amount of knowledge about future green energy production, jobs' characteristics, and the datacenter workload. We have implemented these policies in a GreenPar prototype framework, and evaluated this implementation in Parasol, a solar-powered micro-datacenter we built at Rutgers University [16]. Our evaluation uses workloads comprising jobs executing a subset of the NAS Parallel Benchmark (NBP) suite [35]. The workloads are constructed to resemble traces of real HPC workloads obtained from the Grid Workload Archive [21] and the Parallel Workload Archive [14]. The NBP applications are linked with the MPICH2 MPI implementation [8], and run inside virtual machines (VMs) executing on Xen [7]. (Prior work has demonstrated that virtualization degrades the performance of HPC applications by less than 4% [24, 26].) The virtualization layer allows GreenPar to dynamically change a job's resource allocation, i.e., the set of physical servers hosting the job's VMs, by migrating and consolidating VMs within the datacenter as appropriate, without requiring changes to the MPI implementation.

For repeatability, we also evaluate GreenPar using properly scaled-down traces of green energy production from a solar farm at Rutgers University. When allowing GreenPar to slow jobs' executions down by at most 10%, we show that GreenPar can reduce brown energy consumption by 10% while *reducing* average runtime by 13%, compared to a baseline policy not aware of on-site green energy production. In addition, GreenPar satisfied the jobs' performance SLAs in all our experiments. These results show that while GreenPar may slow down some jobs (within their performance SLAs) to increase brown energy savings, it successfully leverages green energy to speedup overall executions. If applications have good speedup profiles (e.g., close to linear speedup), GreenPar can reduce brown energy consumption by 15% while reducing average runtime by 40%.

In summary, we make the following contributions: (1) we introduce GreenPar, a scheduling framework for parallel HPC workloads in datacenters partially powered by solar energy; (2) we introduce four scheduling policies for GreenPar, each assuming a different amount of knowledge about future green energy production, job characteristics, and datacenter workloads; (3) we implement GreenPar and evaluate it on Parasol, a real solar-powered micro-datacenter; and (4) we present extensive evaluation results for GreenPar, isolating the impact of the different amounts of knowledge in the four policies, and exploring its sensitivity to various parameters.

2. RELATED WORK

As far as we know, GreenPar is the first resource scheduler for green datacenters running parallel HPC applications. Though prior works have considered moldability, malleability, and folding in these applications (e.g., [12, 42]), none of them considered green energy and using it to reduce brown energy consumption and speed up executions. Other works [16–18, 29, 32] considered green energy in the context of scheduling batch applications, but did not dynamically change server allocations of running jobs or attempted to speed up executions using green energy while considering application speedup characteristics.

We divide the other works that relate to GreenPar into two areas: brown power management in parallel applications, and green energy management in datacenters.

Brown power management in parallel applications. Researchers have used multiple techniques to minimize energy consumption or to keep power consumption within a budget when running parallel applications. A popular technique is Dynamic Voltage and Frequency Scaling [25, 38]. Researchers also addressed power measurement and/or modeling of parallel applications [15, 33, 37] in power-aware clusters. Other researchers [9, 36] focused on scheduling for greater efficiency, which indirectly reduces energy consumption. Unlike GreenPar, these works did not consider on-site generation of green energy, where sometimes consuming more (green) energy is a good idea. Leveraging green energy, GreenPar can conserve brown energy even when it is not allowed to trade off performance (i.e., when the SLA does not allow for any performance slowdown).

Green energy management in datacenters. Stewart and Shen [41] and Le et al. [30] have explored request distribution policies for interactive online services that span multiple datacenters to explicitly manage the consumption of green and brown energy. Aksanli et al. [3] have considered the scheduling of mixed batch and service workloads in green datacenters. Akoush et al. [2] proposed workload distribution in virtualized systems. Several efforts have investigated load migration within a datacenter to better leverage on-site green energy production [23, 27, 31, 34].

Unlike these works, GreenPar manages the green energy usage to reduce the brown energy consumption and improve the performance of parallel HPC applications. Moreover, GreenPar dynamically re-allocates resources across running jobs while considering their speedup characteristics to maximize the benefit of green energy.

3. GREENPAR

We propose GreenPar, a job scheduler for parallel HPC workloads running in green datacenters partially powered by solar energy (see Figure 1). Note that, except for the prediction of near-future production of solar energy, *GreenPar is directly applicable to wind-powered green datacenters.* In this section, we first overview GreenPar, and then discuss four specific green-energy-aware policies for it. We also present a simple green-energy-unaware policy that we use as a basis for comparison in Section 5.

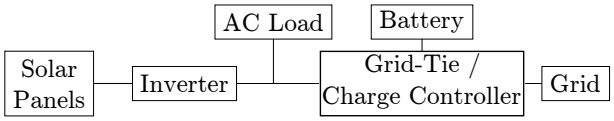

Figure 1: A possible setup for a datacenter partially powered by solar energy. The datacenter is connected to the electrical grid, which can meet the datacenter's peak power demand even when no solar energy is being produced. Batteries are only used as backup for grid outages.

3.1 Overview

GreenPar seeks to maximize green energy consumption and minimize brown energy consumption, while respecting a performance SLA. GreenPar achieves its goals by dynamically changing the resource allocations of jobs based on the availability of green energy, jobs' characteristics, and jobs' accumulated resource allocations.

We assume that each parallel job executes inside a set of VMs (one process per VM), as has been done in previous papers (e.g., [24,26]). GreenPar can dynamically change the number of physical servers assigned to run a job, migrating and consolidating VMs within the datacenter as necessary. Unused servers are put into a low-power state (e.g., ACPI S3) to conserve energy. Note that *GreenPar does not require a virtualized environment.* Rather, it only requires that jobs can adapt to dynamically changing numbers of servers. Bag-of-tasks applications, for example, are amenable to changes in the number of servers without any need for virtualization. Moreover, several papers (e.g., [12, 42]) have demonstrated moldable and malleable HPC frameworks based on MPI, again without using virtualization.

Each parallel job j is submitted to GreenPar with a requested number of servers (N_j) and a maximum acceptable slowdown factor (F_j). Three of the GreenPar policies also ask the user to supply a speedup profile $SP_j(n)$, defined as $SP_j(n) = RT_j(1)/RT_j(n)$, where $RT_j(n)$ is j's runtime when running its VMs on a constant allocation of n servers. (Two policies actually require two speedup profiles, as we explain below.) F_j is a factor ≥ 1 such as 1.1, which would allow GreenPar to slow down job execution by at most 10%. For simplicity, we assume that F_j is the same for all jobs, although our policies can easily be extended to handle different slowdown factors for different jobs.

GreenPar avoids excessively degrading execution times (i.e., degrading them by more than a factor of F) as a result of trying to conserve brown energy. To avoid potentially increasing waiting times due to brown energy conservation, GreenPar reverts back to the user-requested resource allocations when utilization becomes so high that jobs cannot be started immediately upon arrival. As Section 5 demonstrates, GreenPar tends to reduce average runtimes compared to green-energy-oblivious schedulers, which makes it less likely that jobs have to wait at all.

In our current implementation, three GreenPar policies start either twice or four times as many VMs as the requested number of servers. (The other GreenPar policy and the baseline policy always start two VMs per requested server.) The choice between these two options depends on the expected amount of green energy. For example, these policies typically start 16 VMs for $N_j = 8$ in our evaluation environment, where each server has a dual-core processor and so can efficiently host 2 VMs. But they may start 32 VMs, if they are expecting plentiful green energy, so that they are likely to allocate more than N_j servers to job j. Two of these policies use speedup profiles to make decisions, and so require two speedup profiles–one for 16 VMs and another for 32 VMs in our example–for each job, because the job's speedup depends on its number of VMs.

GreenPar then periodically computes the resource allocations for the active jobs. It dynamically increases the numbers of servers allocated to jobs that can achieve further speedups when expecting excess green energy. It dynamically decreases resource allocations of jobs that have accumulated or can be expected to accumulate performance slack (via larger resource allocations than requested when green energy was/will be available) to conserve energy when brown energy is needed.

GreenPar considers the speedup profiles of active jobs when making resource allocation decisions. For example, suppose two jobs, A and B, are each running on 10 servers, and A will not speedup if given more than 10 servers but B will until its allocation exceeds 25 servers. Then, if GreenPar expects that enough green energy will be produced to support 30 servers for a period of time, it will use the speedup profiles to increase B's server allocation to 20, while leaving A's at 10. Note that if B finishes before the green energy production decreases, GreenPar will have successfully used the excess green energy to decrease B's runtime.

However, if both A and B continue to run until a time period when no green energy is available, GreenPar will reduce both their allocations to below 10 servers to conserve brown energy. Of course, GreenPar cannot reduce the allocations so much that either A or B would take more than F times longer to complete compared to when each is running on a constant allocation of 10 servers. In this example, GreenPar may be able to reduce the allocation of B more than A, since B had accumulated some performance slack while running on 20 servers.

While the use of jobs' speedup profiles allows GreenPar to make intelligent decisions about resource allocation, it does make GreenPar reliant on having this information. Speedup information can be gathered across multiple runs of an application, and/or constructed using profiling (e.g., [15, 39]). These approaches can produce very accurate speedup profiles, because HPC applications are often run many times with similar inputs. It is also possible to measure speedup at runtime with sufficient hardware and system software support [36]. Regardless, *to assess the impact on GreenPar if this information was not available, one of the GreenPar policies only uses a hint from the user on whether the submitted job would speedup if given more resources than requested.* Users that are unsure about their applications' behaviors can just supply a "no, the job will not speedup" hint.

3.2 Policies

We now present the four GreenPar policies and the baseline policy. Each GreenPar policy assumes a different amount of knowledge about future green energy production, jobs' characteristics, and future load. All policies divide time into discrete epochs for computing resource allocation schedules. All policies respond to job arrivals and completions within a time epoch.

Reactive. As its name suggests, Reactive does not assume any knowledge or prediction of future job arrivals and runtimes. Instead, it tracks the resource allocation for each job,

```
0.  For each job j, the user specifies the desired number of servers N_j and speedup profile S_j(n)

1.  For each event in {Job Arrival, Job Finish, Change in Green Energy, Reallocation Needed}
2.      For each job j,
3.          Calculate the minimum server allocation n_j for j (lines 14-20 below)
4.      Let C = sum of all n_j
5.      Let G = number of nodes that can be powered by the expected available green energy
6.      For each job j, allocate n_j servers to j
7.      While G > C:
8.          For each job j,
9.              Calculate the decrease in efficiency for further node allocation δ_j
10.         Sort the jobs in ascending order of δ_j
11.         Increase n_j by a threshold number of servers m_j to the first job j in the list
12.         C = C + m_j and update δ_j
13.         Resort the jobs in ascending order of δ_j

14. Calculate Minimum number of nodes for job j:
15.     if j is new, average speedup as_j = 0 and achieved runtime rt_j = 0
16.     Assume that j will run for another threshold time period t
18.     n_j = the minimum number of nodes such that as_j ≥ F · S_j(N_j)
19.     Schedule a Reallocation Needed event t time into the future
20.     Return n_j
```

Figure 2: Pseudocode for the Reactive scheduling algorithm.

predicts the green energy production for the next time epoch at the end of every time epoch, and reacts to the expected availability of green energy. When there is excess green energy, Reactive tries to increase the server allocation of active jobs to decrease runtimes. It greedily tries to increase the allocation of jobs with the best marginal speedups first, seeking maximum reduction in job runtimes for each additional allocated server. Subsequently, when there is insufficient green energy, Reactive tries to shrink the server allocations of active jobs that have accumulated some performance slack to conserve brown energy. It again greedily tries to reduce the allocations of jobs with the smallest marginal speedups first, seeking a minimum increase in runtimes for each removed server.

Reactive does not need to know jobs' runtimes to respect the SLA. Instead, knowing the requested number of servers and jobs' speedup profiles is sufficient since:

$$MaxRunTime = F \cdot RunTime_{j,1}/SP_j(N), \text{ and}$$
$$RunTime = RunTime_{j,1}/AverageSpeedup_j$$

implying that we can meet the performance SLA if:

$$AverageSpeedup_j \geq SP_j(N)/F$$

In our current implementation, Reactive always starts a job with twice as many VMs as the number of requested servers, so it requires only one speedup profile for each job. Figure 2 shows the pseudo-code for Reactive.

Offline optimization. For comparison, we develop an optimization-based policy with oracular knowledge of job arrivals, job speedups and runtimes, and green energy production. We call this policy Offline because GreenPar is unlikely to have the assumed information.

Table 1 lists the parameters of our optimization framework. Using them, we formulate the optimization problem shown in Figure 3. The optimization works on a scheduling horizon (e.g., 24 hours into the future) that is divided into discrete time epochs (t). Solving the optimization problem produces V_j (the number of VMs to run job j) and an epoch-based schedule of the number of servers ($S_j(t)$) assigned to each job j arriving in the scheduling horizon. Our current implementation considers starting each new job with twice or four times as many VMs as the user-requested number of servers for the job. The optimization is re-solved periodically to account for new knowledge (e.g., jobs arriving after the current 24-hour scheduling horizon).

Table 1: Framework parameters. Time epochs in the scheduling horizon are numbered from 1 to T.

Symbol	Meaning
T	Set of time epochs comprising scheduling horizon
J	Set of jobs arriving in the scheduling horizon
β	Relative weight of the average runtime vs. the brown energy cost in the objective function
F	Maximum acceptable slowdown factor
C	Number of servers in the datacenter
$GE(t)$	Predicted green energy production in epoch t
$PB(t)$	Price of the brown energy during epoch t
A_j	Epoch that job j arrives and starts running
E_j	Epoch that job j completes
N_j	Number of servers requested for job j
$RT_{j,1}$	Runtime of job j when running on 1 server
$SP_j(n,v)$	Job j's speedup when running on n servers with v VMs
$ME_j(n)$	Migration energy when job j's server allocation changes by n
$Pow_j(n)$	Power demand of job j when running on n servers
V_j	Number of VMs used to run job j
$S_j(t)$	Number of servers allocated to job j in epoch t

This optimization minimizes $Objective$, a weighted sum of the brown energy cost ($BrownCost$) and average job runtime ($AvgRuntime$).[3] The β factor allows us to adjust the relative importance of conserving brown energy versus increasing job runtimes. $BrownCost$ is a function of the brown energy use ($BrownEn(t)$) over time and the brown energy price. Brown energy is consumed only when more energy ($En(t)$) is needed than the amount of green energy being produced. Energy is consumed both by jobs executing and migrations required to adapt to dynamically changing resource allocations ($MiEn_j(t)$). We use the v parameter in $SP_j(n,v)$ to represent the two user-provided speedup curves.

GreenPar solves the optimization problem under constraints formulated to ensure that the jobs' performance SLAs are met, jobs complete their executions, jobs are not preempted once they start running, and the allocated servers do not exceed the datacenter capacity.

The optimization problem can be transformed into a linear problem except for: (a) the migration energy $ME_j(n)$

[3]Previous studies have used different optimization objectives, e.g., $Energy \times Delay$ or $Energy \times Delay^2$. We choose this linear combination because it has been used successfully [4], and it allows us to formulate a linear optimization problem that can be solved efficiently.

$$Objective = BrownCost + \beta \cdot AvgRuntime \quad (1)$$

$$BrownCost = \sum_{t \in T} BrownEn(t) \cdot PB(t) \quad (2)$$

$$BrownEn(t) = \begin{cases} En(t) - GE(t) & \text{if } En(t) > GE(t) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$En(t) = \sum_{j \in J} (Pow_j(S_j(t)) \cdot |t| + MiEn_j(t)) \quad (4)$$

$$MiEn_j(t) = ME_j(|S_j(t) - S_j(t-1)|) \quad (5)$$

$$AvgRuntime = \frac{1}{|J|} \cdot \sum_{j \in J} \sum_{t \in T} Running_j(t) \cdot |t| \quad (6)$$

$$Running_j(t) = \begin{cases} 1 & \text{if } S_j(t) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

$$E_j = A_j + \sum_{t \in T} Running_j(t) \quad (8)$$

$$AvgSpeedup_j = \sum_{t \in T} SP_j(S_j(t), V_j)/(E_j - A_j + 1) \quad (9)$$

Figure 3: Optimization framework.

and the power demand of a job $Pow_j(n)$ may be non-linear functions, and (b) the speedup functions $SP_j(n, v)$ may be non-linear. We address the first issue by experimentally measuring and using approximate linear functions for both functions; in fact, we use only a single $Pow(n)$ and $ME(n)$ function for all jobs. In our evaluations, we found that this assumption introduced only small inaccuracies. We address the second issue by instantiating the speedup functions of each job as a table look-up using Integer Programming. Thus, the result is a Mixed Integer Linear Programming (MILP) problem that we can efficiently solve (at the sizes we consider) with standard solvers (e.g., [22]).

Aggressive. This policy is essentially the same as Offline except that it only assumes knowledge of jobs' runtimes and speedup profiles. It predicts future green energy production, implying that $GE(t)$ includes prediction errors. We discuss green energy predictions further in Section 4. It also solves the optimization problem only for active jobs, i.e., J only contains the jobs that have already arrived, and thus does not rely on knowledge of future arrivals.

Aggressive re-solves the optimization whenever a new job arrives. The optimization generates the number of VMs (V_j) for the new job and an epoch-based schedule of the number of servers ($S_j(t)$) assigned to each job j. Aggressive must also re-solve the optimization when there are significant changes to the predictions of green energy production. In our experiments, solving the Aggressive optimization problem takes less than 2 seconds.

Nebulous. Finally, we develop and study Nebulous, a policy that does not assume accurate knowledge of speedup profiles. Rather, it uses a user-provided hint about job speedup (excellent, good, fair, poor) and a rough estimate of job runtime. With these hints, it is impossible for Nebulous to support the same performance SLA as the previous policies. Thus, Nebulous instead seeks to ensure that the average resource allocation to each job is never less than $1/F$ times the number of requested servers.

Nebulous predicts green energy production, and, when a job arrives, it starts the job with more VMs if it expects that there will be enough green energy to run the larger number of VMs for the estimated duration of the job. This approach is conservative in that the runtime estimate is for the smaller number of VMs, so it is likely to be longer than the runtime when using the larger number of VMs.

Nebulous is then somewhat similar to Reactive. When there is excess green energy, it greedily allocates more servers to jobs with the best speedup hints and smallest average allocations thus far vs. their requested numbers of servers. Using the intuition that marginal speedup typically decreases with increased resource allocations, it only gives the smallest number of extra servers possible to each job before considering the next job. When green energy production decreases, Nebulous reduces server allocations to jobs in the reverse order (i.e., take away servers from jobs with poor speedup hints and most accumulated slack first).

Finally, since Nebulous only uses the runtime estimate to decide whether to run a job with more VMs, it can use statistics from previous runs of the application. When an application is run for the first time, Nebulous conservatively assumes that the job will run for a long time, implying that it does not start the job with more VMs. In our evaluations, we use the average of the previous runs of each application and correct speedup hints.

Baseline. We use a Baseline policy as a basis for comparison in our evaluation of the green-energy-aware policies. Baseline runs each job as it arrives, starting each job with twice the number of VMs as requested servers.

4. EVALUATION METHODOLOGY

Prototype implementation and evaluation platform. We have implemented a prototype of GreenPar comprising roughly 2200 lines of python code. The prototype includes all four policies Reactive, Aggressive, Offline, and Nebulous.

We evaluate the prototype running on Parasol. Our evaluation uses 55 servers, where each server is equipped with a dual-core 1.6GHz Atom processor, 4GB of memory, one 250GB hard disk, and one 64GB solid-state drive. This cluster uses 5 machines as NFS servers to hold VM images and application data, and the remaining 50 machines for running jobs executing inside VMs. Each server consumes between 22W to 30W, giving a peak power consumption of 1.5kW for the computing servers. The servers are interconnected with a 1Gbps Ethernet network.

The 50 compute servers run Xen 4.3 [7], using the Linux 3.11 kernel for both the host and guest OSes. Each VM image is configured with 1 virtual CPU, 384MB of memory, and a 4GB disk. Baseline always assigns two VMs to each server, giving each VM a dedicated core. The other policies can assign 1 VM to each server when a job is given extra resources, or up to 4 VMs per server when the job is consolidated to save energy. Although running a VM with just one virtual CPU on a dual-core server may seem somewhat inefficient, it can speedup execution by 66% over running 2 VMs on the server for communication-intensive applications (e.g., NAS IS). We use Xen's live migration capability to migrate VMs without interrupting job executions. The virtual disks are stored on the NFS servers, and so do not have to be migrated. Measurements show that live migrations had very little impact on job runtimes. Idle servers are placed into the ACPI S3 state to conserve energy.

Parasol allows us to measure the energy consumption of individual servers. Thus, all results below are from actual measurements of energy consumed by executing workloads.

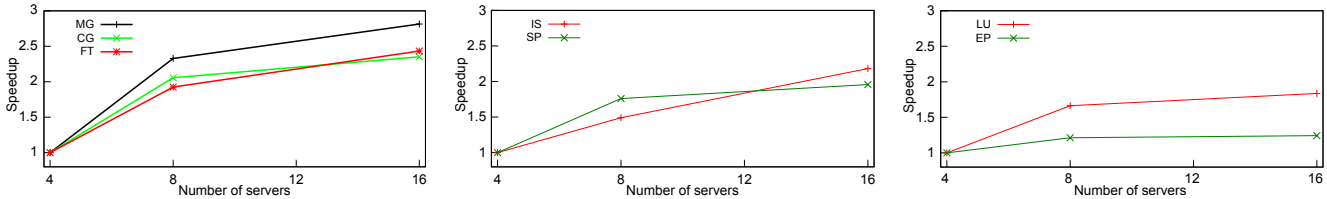

Figure 4: Speedup profiles of NPB applications running with 16 VMs on different numbers of physical servers. The speedup profile of BT is not shown because it is exactly same as SP.

Table 2: Core count mapping for Intrepid.

Intrepid core count	256, 512	1024, 2048	4096	Others
Mapped core count	4	8	16	32

Workloads. We study two workloads created to emulate two real traces. The first workload, called Grid5k, emulates a trace from the Grid Workload Archive [11]. The original trace was collected on the Grid'5000 system [21], a 2,218-node distributed system spread across 9 sites in France, from May 2004 to November 2006. We chose an arbitrary 24-hour period from this trace and filtered out short running jobs that ask for just one processor core, assuming that they are small test runs, to scale the workload to our datacenter. After filtering, the chosen sub-trace has 26 jobs, with a peak processing demand of 96 cores. Most of the results presented below were obtained using this workload trace.

The second workload, called Intrepid, emulates a trace from the Parallel Workload Archive [14]. The original trace was collected on the Intrepid system, a 40-rack, 40,960-node Blue Gene/P system deployed at Argonne National Laboratory, from January 2009 to August 2009. We selected an arbitrary 24-hour portion of the trace, and then scaled down job sizes as explained in the next paragraph to fit our datacenter. The selected period has 113 jobs, with a peak processing demand of 86 cores.

Each of the workload traces includes information on job arrival time, job runtime, and the number of cores requested. However, they do not include information about the applications nor the input data. Thus, we use a subset of applications from the NAS Parallel Benchmark (NPB) [35] to instantiate the workloads. NPB contains 8 applications with different input sizes. We use a mix of applications and input sizes to emulate the two different traces. Figure 4 shows the speedup curves for all the applications used in our workload.

Most NPB jobs require a power-of-2 number of VMs. We map each job in the Grid5k trace to the closest NPB application in terms of core count (assuming 1 VM per core) and runtime, with 2 VMs running on each dual-core server. With this mapping, the resulting Grid5k workload includes 12 FT, 6 LU, 3 MG, 2 CG, and 1 each of SP, EP and IS. In contrast, Intrepid jobs ask for large numbers of cores, with the minimum being 256. To scale these jobs to our datacenter, we map the Intrepid jobs following Table 2. The resulting workload contains 73 IS, 24 MG, 14 CG, 1 FT and 1 SP applications. Figure 5 shows the workload demand (in number of cores) of Grid5k and Intrepid, as a function of time. The workloads never require more servers than are available (50 servers with 100 cores) in our setup.

Energy prices. We use the average brown energy price for New Jersey: 10.55 cents/kWh [13]. The use of a constant price equates saving brown energy with reducing brown energy cost. However, in general, both Aggressive and Offline can leverage dynamically changing energy prices to reduce

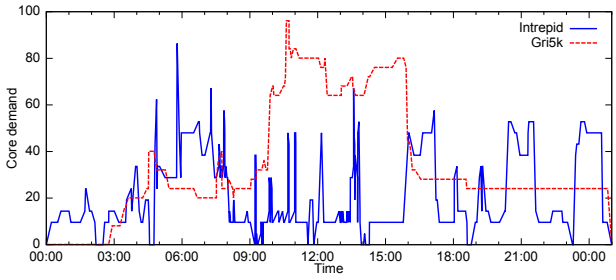

Figure 5: Workload demand in number of cores, as a function of time.

the brown energy cost. We assume self-generation of solar energy, so that green energy has zero (marginal) cost. Our framework can be easily extended to account for a non-zero green energy cost, if desired.

Optimization weighting factor. The outcome of the Aggressive and Offline policies is affected by the value of β. β controls the relative importance of runtime compared to electricity cost. To gauge this relationship, we first convert runtime to a cost figure (using the cost of computation in the cloud) and then compare it to the electricity cost of the same computation and for the same runtime. Specifically, we compare the per-hour cost of renting a large VM instance on Amazon's EC2 [5] ($0.24) with the per-hour electricity cost of a server with the same specification as the instance (roughly 0.3kWh × $0.1055/kWh). So, our chosen value for β is 8 ($= \frac{\$0.24}{0.3kWh \times \$0.1055/kWh}$).

We also ran experiments with β ranging from 0.8 to 80. We omit these results as they were as expected, and do not affect the observations in our evaluation.

Estimating migration overheads. We measured the migration time for all the applications in our environment. The measured migration time (and thus energy overhead) does not vary much among the applications. So, we use the average migration time for every application when solving the optimization problem in Aggressive and Offline.

Solar energy production and prediction. Although Parasol is partially powered by a set of solar panels, for repeatability, we used traces of solar energy production in our experiments. However, we have run many validation runs on Parasol, using real predictions and production of solar energy. We show results from one of these runs below to show that GreenPar behaves as expected under live execution.

In our experiments, we model the solar power generation as a scaled-down version of a solar farm at Rutgers that has a rated production capacity of 1.4MW. We scale the farm's production down to 8 solar panels capable of producing 1.88kW. After derating, the peak solar power production matches the peak power consumption of our datacenter.

We evaluate our scheduling policies using three days with different amounts of solar energy production: one sunny

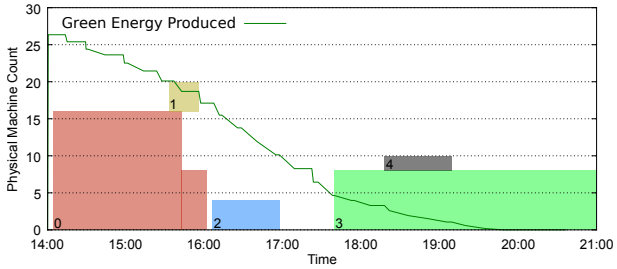

Figure 6: Green energy production and server allocations during a 7-hour validation run of Grid5k.

day (5/9/11) with high solar energy production (totaling 12kWh), one day (6/16/11) with medium solar energy production (totaling 8kWh), and one day (5/15/11) with low solar energy production (totaling 2.84kWh). We call these the "High", "Medium", and "Low" days, respectively.

We use the method from [18] (which is based on [40]) to predict solar energy production for a horizon of 48 hours. Briefly, the model relates solar energy generation to cloud cover as $E_p(t) = B(t)(1 - CloudCover)$, where $E_p(t)$ is the amount of energy predicted for time period t, $B(t)$ is the amount of energy expected under ideal sunny conditions, and $CloudCover$ is the forecasted percentage cloud cover. We use Intellicast.com's weather forecasts, which predicts $CloudCover$ for each hour of the next 48 hours. This leads to prediction granularity (i.e., t) of one hour. We set $B(t)$ for each hour of the day to the amount of energy generated during that hour on the day with the highest energy generation from the previous month.

Of course, weather forecasts are sometimes wrong. Thus, the prediction includes a technique for using recent energy production to predict future production when predictions based on weather forecasts are inaccurate [17]. The evaluation in [18] shows that one-hour ahead predictions using this method produce inaccuracies of around 11%.

Our prediction method produces different accuracies for the High, Medium, and Low days. Predictions are mostly accurate for the High day, although there are some under-predictions. They are also mostly accurate for the Low day. However, those for the Medium day contain significant errors, with periods of over- and under-predictions.

Accelerated experiments. To make it possible to run a large number of experiments, each experiment is an accelerated run of the workload. Specifically, the time frame is accelerated by a factor of 60, making 1 minute of experiment execution represent 1 hour of real time. We accelerate jobs' execution times by reducing the size of the inputs. We also constrained the system to not migrate VMs more often than once a minute (1 hour in real time). Note that this is pessimistic in two ways: (a) the migration time did not scale down linearly; thus, migration overheads would have been smaller in un-accelerated execution; and (b) lower migration overheads would have allowed our policies to adjust allocations more often (e.g., every 15 minutes). The validation experiment shown in Figure 6 was run in real time (that is, it was not accelerated).

Experimental setup. We set the scheduling horizon to 24 hours, and divide this horizon into 15-minute epochs.[4] We use the Gurobi solver [22] to solve the optimization problems for Aggressive and Offline. In our experiments, solutions of Aggressive take less than 2 seconds to execute. Offline runs completely offline to determine the best case for comparison, so we let it execute for 30 minutes in our experiments.

We use the same amount of maximum acceptable runtime slowdown for all jobs for Reactive, Offline, and Aggressive. The default is 10%, although we also study settings in the 0%–50% range. For Nebulous, we use a maximum reduced average resource allocation of 10%. In our experiments, *GreenPar is able to satisfy all performance SLAs.*

Finally, to make efficient use of resources, GreenPar only allocates powers-of-2 servers to each job, so that the number of VMs per server for a job is always the same. Also, GreenPar cannot allocate more servers to a job than the number of VMs in that job, and cannot consolidate more than 4 VMs onto a single server. All of these limit the flexibility with which GreenPar can adjust jobs' server allocations, but are limitations of our experimental setup rather than fundamental to GreenPar itself. GreenPar would perform better if jobs could dynamically adapt to arbitrary server allocations.

5. VALIDATION AND EVALUATION

In this section, we first present a validation experiment of GreenPar running on Parasol with real energy production and prediction. We then present our evaluation of the four GreenPar policies using our scaled-down solar traces.

Validation. Figure 6 shows the results from a validation experiment using Aggressive and the Grid5k workload. The experiment ran over 7 hours on Parasol, using real solar energy production and live predictions of solar energy production. In Figure 6, each box shows the server allocation of a job over its lifetime. The green line depicts the green energy available over time. The numbers in the colored boxes are the job identifiers. In this execution, jobs 0, 1, and 2 requested 8, 2, and 2 servers, respectively. However, because of the high green energy availability, GreenPar doubled the allocations of each of these jobs to reduce their runtimes. When the production of green energy decreased toward the end of job 0's execution, GreenPar adjusted by reducing job 0's server allocation to 8 servers. GreenPar was able to reduce the runtimes of these jobs by an average of 23.8%.

Evaluation. Figure 7 compares the brown energy consumption, green energy usage, total number of VMs, and average runtime reduction achieved by Baseline and the four GreenPar policies when running Grid5k for the High day. The total number of VMs is the sum of all VMs used by all the jobs; a larger number of VMs means that jobs were executed with greater parallelism. The brown energy consumption, green energy usage, and total number of VMs are normalized compared to those of Offline (Y-axis on the left). Average runtime reduction is the percentage reduction of jobs compared to when they are scheduled by Baseline (Y-axis on the right).

These results show that all four policies can reduce brown energy consumption *and* reduce job runtimes. They achieve these savings by slowing down job execution when only brown energy is available, and speeding up job execution when green energy is available. While Reactive only reduces brown energy consumption by a small amount (~2% reduction), it successfully uses more green energy to reduce average job runtimes by 5%. This means that even when the policy has

[4]The epoch duration can involve tradeoffs between scheduling overhead and faster response to changing conditions. However, our evaluation is insensitive to this parameter, since migration overheads are small and near-future prediction of green-energy is relatively accurate.

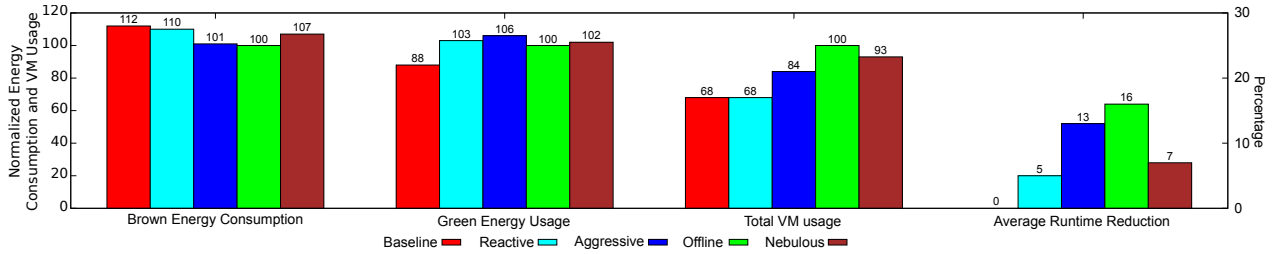

Figure 7: Comparison of policies for Grid5k workload running on the High day.

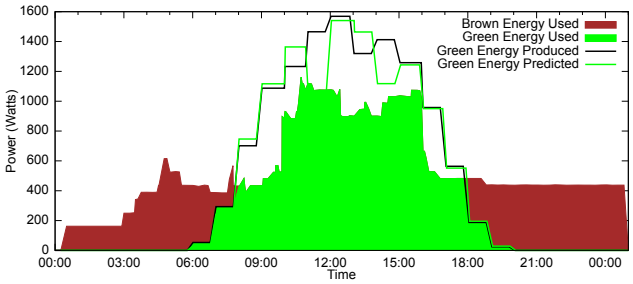

Figure 8: Behavior of Baseline, Grid5k, High day.

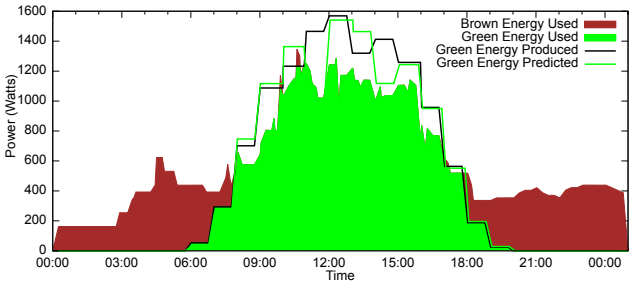

Figure 9: Behavior of Reactive, Grid5k, High day.

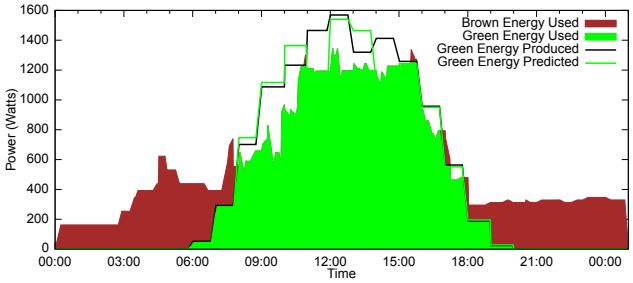

Figure 10: Behavior of Aggressive, Grid5k, High day.

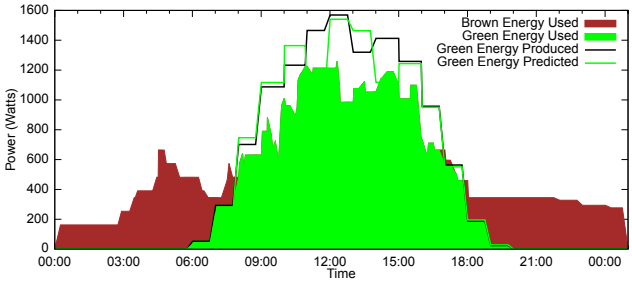

Figure 11: Behavior of Offline, Grid5k, High day.

no knowledge of job arrivals and runtimes, it can leverage green energy to improve runtimes.

Aggressive is better than Reactive because it uses knowledge of job runtimes and predictions of green energy availability in the near future to more aggressively start jobs with higher levels of parallelism; the same set of jobs is run with 84 VMs under Aggressive compared to 68 under Reactive. This allows Aggressive to successfully use more green energy, less brown energy, and achieve shorter job runtimes.

In fact, Aggressive behaves almost as well as Offline. It uses only slightly more brown energy (1% more), and almost matches Offline in the average reduction of runtimes (13% vs. 16%). The advantages of Offline arise from its oracle knowledge of the future. Aggressive immediately runs an arriving job with a larger number of VMs if it expects green energy to be available. However, this may prevent it from running the next arriving job with a large number of VMs–there may be insufficient resources left–even though the subsequent job has better speedup characteristics. Offline can make the correct choice given its knowledge of the future. Interestingly, Offline uses less green energy because it uses it more efficiently; it allocates the energy to the jobs with the best speedup, allowing these jobs to finish more quickly and stop consuming energy.

Finally, Nebulous behaves slightly better than Reactive even though it is given only hints about speedup. This is because it aggressively starts jobs with more VMs when expecting excess green energy in the future. Thus, Nebulous uses more green energy to speedup jobs than Reactive.

On the other hand, Nebulous cannot match Aggressive because its heuristic for choosing the number of VMs is less accurate. In our implementation, Nebulous only chooses the larger number of VMs (per requested server) if it expects to have excess green energy for the entire estimated runtime of a job. However, the job may finish much faster when it is given extra resources. Thus, Nebulous's decisions are often overly conservative compared to Aggressive. Also, Nebulous's allocations of extra resources between multiple jobs are often not as good as Aggressive's allocations, because it does not have accurate speedup information.

Detailed power demand profiles for the five policies when running Grid5k for the High day are shown in Figures 8–12. The first 6 hours are similar for all policies because there is no green energy production, and the jobs can only be slowed down by a maximum of 10%. In the daytime, when green energy becomes available, the green-energy-aware policies increase power consumption by either increasing the server allocations of active jobs (all four policies) and/or starting new jobs with more VMs, allowing larger than requested server allocations (Aggressive, Offline, and Nebulous). As mentioned earlier, Offline consumes less green energy even though it runs jobs with more parallelism (greater number of VMs) because it allocates the green energy to the most efficient jobs, leading to the greatest reduction in job runtimes. Finally, Aggressive and Offline can reduce brown energy consumption at night (right portion of the figures) either because jobs finished faster (they were run with more parallelism when green energy was available) or enough per-

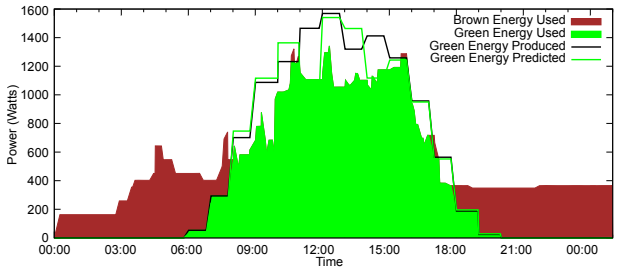

Figure 12: Behavior of Nebulous, Grid5k, High day.

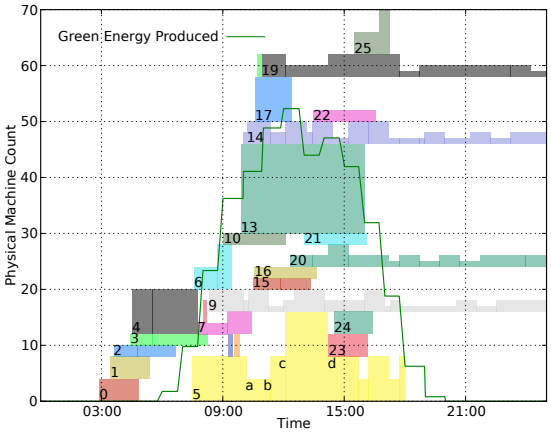

Figure 13: Aggressive's server allocation for Grid5k.

formance slack has been accumulated so that some jobs can be consolidated onto fewer physical machines compared to Baseline. Reactive and Nebulous achieve smaller savings.

For a more detailed look at the behavior of a green-energy-aware policy, Figure 13 shows the allocation of physical servers to jobs for Aggressive over time. The green line, boxes, and numerical labels are the same as for Figure 6. A closer look at job 5 reveals several interesting points. First, Aggressive starts job 5, which requested 4 servers, with 16 VMs because it is expecting to have excess green energy. This allows Aggressive to run job 5 on more servers, consuming green energy to speedup job execution. Initially, Aggressive allocates 8 servers to the job (2 VMs per dual-core machine). At point 'a', job 14 arrives with better speedup characteristics. Thus, Aggressive shrinks job 5's server allocation to 4. Later, as more green energy becomes available, Aggressive increases job 5's allocation to 8 (point 'b') and then to 16 (point 'c'). Note that, with 16 servers, job 5 is being run with only 1 VM per server. This allocation achieves limited speedup since the second core on each server is frequently idle. However, since the green energy is "free" in this case, it is beneficial to use this energy whenever some speedup is still possible. As green energy production decreases over the day, and other jobs enter the system, Aggressive again decreases job 5's allocation (point 'd' and beyond). In comparison, job 5 ran for a much longer time with a static allocation of 4 servers under Baseline (not shown here).

Impact of green energy availability. We also run Baseline and Aggressive for the Medium and Low days. Figure 14 shows the behavior of Aggressive for the Medium day. As previously mentioned, this day has significant mis-predictions of the solar energy production, including both under- and over-predictions. Over-predictions can lead Aggressive to use brown energy unnecessarily (and less efficiently). On the other hand, under-predictions can lead it to waste green energy. Overall, Aggressive still reduces the average runtime by 27% despite the mis-predictions. However, it uses 5% more brown energy than Baseline. On the Low day, Aggressive and Baseline behave the same because there was little green energy to leverage.

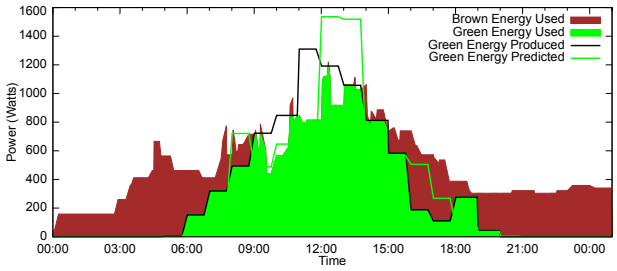

Figure 14: Behavior of Aggressive, Grid5k, Medium day.

We also study Aggressive and Baseline for different green energy production capacities. Scaling the solar power system does not change the overall behaviors. On the High day, a peak solar production capacity of 125% of the peak datacenter power consumption leads to an 11.3% reduction in the brown energy consumption (compared to 10% above), and a 12.9% reduction in the average runtime (compared to 12% above). Scaling down the peak solar energy production to 75% of the peak datacenter power consumption leads to an 8.6% reduction in the brown energy consumption and a 10.3% reduction in the average runtime.

Impact of application speedup. GreenPar's efficacy can be strongly affected by the speedup characteristics of jobs in the workloads. To see the impact of having jobs with better speedups, we constructed a Grid5k workload using only EP jobs. When an EP job is run with 16 VMs, it achieves much better speedup (much closer to linear) than that shown in Figure 4. When running this workload, Aggressive reduces the brown energy consumption by 15% while reducing the average runtime by 40% compared to Baseline.

Impact of performance SLA. When we disallow GreenPar to slow down job execution, i.e., set F to 1, Aggressive can still reduce the brown energy consumption by 8% and reduce the average runtime by 12%. If we instead allow a 50% slow down factor, Aggressive consumes 16% less brown energy than Baseline while reducing the average runtime by 9%. In this case, Aggressive's lack of knowledge about future job arrivals limits its performance. Specifically, before the green energy production starts to ramp up, Aggressive (aggressively) slows down the execution of active jobs, planning to use the green energy produced later on to recoup the performance loss. However, new jobs arriving later in the day will consume some of the green energy, leading Aggressive to consume brown energy instead to catch up; the catching up is typically done at lower efficiency since execution at larger server allocations is typically less efficient.

Impact of runtime mis-estimations. Aggressive outperforms Baseline but relies on knowing jobs' runtimes. We have studied their relative performance where the user provided job runtimes to Aggressive are inaccurate by ±25%. These inaccuracies did not significantly affect Aggressive since it recomputes a new schedule whenever a job completes anyways. Thus, the performance differences between Aggressive and Baseline were close to those presented above.

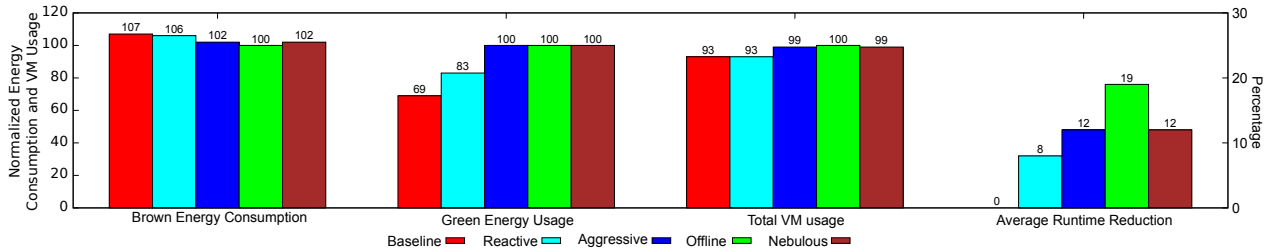

Figure 15: Comparison of policies for the Intrepid workload running on the High day.

Impact of datacenter utilization. Under Baseline, Grid5k produces a datacenter utilization of approximately 33%. The traces we use to construct our workloads exhibit similar utilizations. Nevertheless, we also run experiments with higher utilizations to gauge GreenPar's behavior under heavy load. Specifically, we doubled (2x) and tripled (3x) Grid5k, causing lengthy queue build up over time in Baseline. At 2x load, 12% more jobs complete under Aggressive than Baseline. This is because Aggressive is still occasionally able to give some jobs extra resources, allowing them to complete faster and reduce contention. At 3x load, the two policies perform the same. Thus, we conclude that GreenPar's benefits tail off as utilization becomes very high. *However, GreenPar does not harm performance even at very high loads.*

Impact of workload characteristics. Figure 15 compares the brown energy consumption, green energy usage, total number of VMs, and average runtime reduction achieved by the policies when running Intrepid for the High day. These results show the same trends as those observed for Grid5k. Interestingly however, Nebulous's performance almost matches Aggressive. This is because most jobs are short so that they easily fit within periods of high solar energy. Thus, both policies make similar decisions about the number of VMs to start per job.

6. CONCLUSIONS

In this paper, we proposed GreenPar, a scheduler for parallel HPC workloads running in datacenters partially powered by solar energy. GreenPar seeks to maximize the use of green energy to reduce job runtimes and brown energy consumption. We have implemented and evaluated it using realistic workloads, running on a real solar-powered datacenter. Our results show that GreenPar can increase green energy consumption and decrease brown energy consumption, while reducing average runtime at the same time. The results also show that an online policy using information about job speedups, runtimes, and predictions of solar energy production can come close to matching an offline policy that additionally has perfect information about future job arrivals and green energy production. Finally, GreenPar still provides benefits (albeit smaller ones) when it only has rough hints for speedup and runtime.

We conclude that GreenPar can become an important software component in green datacenters that run HPC workloads, helping to improve the sustainability of our Information Technology ecosystem.

Acknowledgments. We would like to thank William Katsak and the anonymous reviewers for their helpful comments and suggestions. This work was partially supported by NSF grant CSR-1117368 and the Rutgers Green Computing Initiative.

7. REFERENCES

[1] AISO.net. Web Hosting as Nature Intended, 2012. `http://www.aiso.net`.

[2] S. Akoush, R. Sohan, A. Rice, A. Moore, and A. Hopper. Free Lunch: Exploiting Renewable Energy for Computing. In *Workshop on hot topics in operating systems (HotOS)*, 2011.

[3] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing. Utilizing Green Energy Prediction to Schedule Mixed Batch and Service Jobs in Data Centers. In *Workshop on Power-Aware Computing and Systems (HotPower)*, 2011.

[4] S. Albers and H. Fujiwara. Energy-Efficient Algorithms for Flow Time Minimization. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, 2006.

[5] Amazon EC2 Pricing. `http://aws.amazon.com/ec2`, Retrieved on February 2013.

[6] Apple Inc. Apple Environmental Responsibility Report. `https://www.apple.com/environment/reports/docs/apple_environmental_responsibility_report_0714.pdf`, 2014.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Symposium on Operating Systems Principles (SOSP)*, 2003.

[8] A. Bouteiller, F. Cappello, T. Hérault, G. Krawezik, P. Lemarinier, and F. Magniette. MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging. In *Supercomputing (SC)*, 2003.

[9] J. Corbalan and J. Labarta. Improving Processor Allocation Through Run-time Measured Efficiency. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2001.

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

[11] Delft University of Technology. The Grid Workloads Archive. `http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Workloads.Gwa-t-2`.

[12] T. Desell, K. El Maghraoui, and C. A. Varela. Malleable Applications for Scalable High Performance Computing. *Cluster Computing*, 10(3), 2007.

[13] Energy Information Administration. Average Retail Price of Electricity to Ultimate Customers by End-Use Sector, by State. `http://www.eia.gov/electricity/monthly/epm_table_grapher.cfm?t=epmt_5_6_b`, Retrieved on January 2013.

[14] D. Feitelson. Parallel Workload Archive. http://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/index.html.

[15] R. Ge and K. Cameron. Power-aware Speedup. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2007.

[16] I. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini. Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy. In *International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, 2013.

[17] I. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *Supercomputing (SC)*, 2011.

[18] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In *European Conference on Computer Systems (EuroSys)*, 2012.

[19] Green House Data. An Economically Responsible Data Center, 2012. http://www.greenhousedata.com.

[20] GreenQloud, 2013. http://greenqloud.com.

[21] Grid'5000. Grid'5000 Experimentation Platform. www.grid5000.fr.

[22] Gurobi Optimization Inc. Gurobi Optimization. http://www.gurobi.com.

[23] M. E. Haque, K. Le, I. Goiri, R. Bianchini, and T. D. Nguyen. Providing Green SLAs in High Performance Computing Clouds. In *International Green Computing Conference (IGCC)*, 2013.

[24] S. Hazelhurst. Scientific Computing Using Virtual High-performance Computing: A Case Study Using the Amazon Elastic Computing Cloud. In *Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT)*, 2008.

[25] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang. Enhanced Energy-Efficient Scheduling for Parallel Applications in Cloud. In *Symposium on Cluster, Cloud, and Grid Computing (CCGRID)*, 2012.

[26] W. Huang, J. Liu, B. Abali, and D. K. Panda. A Case for High Performance Computing with Virtual Machines. In *International Conference on Supercomputing (ICS)*, 2006.

[27] K. Kant, M. Murugan, and D. H.C.Du. Willow: A Control System for Energy and Thermal Adaptive Computing. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2011.

[28] J. Koomey. Growth in Data Center Electricity Use 2005 to 2010, 2011. Analytic Press.

[29] A. Krioukov, C. Goebel, S. Alspaugh, Y. Chen, D. Culler, and R. Katz. Integrating Renewable Energy Using Data Analytics Systems: Challenges and Opportunities. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, March 2011.

[30] K. Le, O. Bilgir, R. Bianchini, M. Martonosi, and T. D. Nguyen. Capping the Brown Energy Consumption of Internet Services at Low Cost. In *International Green Computing Conference (IGCC)*, 2010.

[31] C. Li, A. Qouneh, and T. Li. iSwitch: Coordinating and Optimizing Renewable Energy Powered Server Clusters. In *International Symposium on Computer Architectur (ISCA)*, 2012.

[32] C. Li, R. Wang, T. Li, D. Qian, and J. Yuan. Managing green datacenters powered by hybrid renewable energy systems. In *International Conference on Autonomic Computing (ICAC)*, 2014.

[33] D. Li, B. De Supinski, M. Schulz, K. Cameron, and D. Nikolopoulos. Hybrid MPI/OpenMP power-aware computing. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2010.

[34] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and Cooling Aware Workload Management for Sustainable Data Centers. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2012.

[35] NASA Advanced Supercomputing Division, Retrieved on December 2013. www.nas.nasa.gov/publications/npb.html.

[36] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Using Runtime Measured Workload Characteristics in Parallel Processor Scheduling. In *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 1996.

[37] A. Porterfield, S. Olivier, S. Bhalachandra, and J. Prins. Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs. In *International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, 2013.

[38] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz. Bounding Energy Consumption in Large-Scale MPI Programs. In *Supercomputing (SC)*, 2007.

[39] M. Shantharam, Y. Youn, and P. Raghavan. Speedup-Aware Co-Schedules for Efficient Workload Management. *Parallel Processing Letters*, 23(02), 2013.

[40] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy. Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems. In *International Conference on Sensor Mesh and Ad Hoc Communications and Networks (SECON)*, 2010.

[41] C. Stewart and K. Shen. Some Joules Are More Precious Than Others: Managing Renewable Energy in the Datacenter. In *Workshop on Power Aware Computing and Systems (HotPower)*, 2009.

[42] G. Utrera, J. Corbalan, and J. Labarta. Implementing Malleability on MPI Jobs. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2004.