Design and Evaluation of an Energy Agile Computing Cluster

Andrew Krioukov
Sara Alspaugh
Prashanth Mohan
Stephen Dawson-Haggerty
David E. Culler
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-13
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-13.html

January 17, 2012

Design and Evaluation of an Energy Agile Computing Cluster

Andrew Krioukov, Sara Alspaugh, Prashanth Mohan, Stephen Dawson-Haggerty,
David Culler, Randy Katz
University of California, Berkeley
$\{$*krioukov, alspaugh, prmohan, stevedh, culler, randy*$\}$*@eecs.berkeley.edu*

Abstract

Variable, intermittent renewable sources of energy are being introduced into the national electric grid at scale. However, the current grid paradigm of load-following supplies is unable to utilize these sources without impractical deployments of backup generation and grid-scale energy storage. Thus, it will be crucial to have energy agile loads: electric loads that can dynamically adapt their energy consumption. Data centers are prime candidates for becoming energy agile because they are large energy consumers and often have workloads that can be shaped to vary their power consumption. In this paper we present an energy agile cluster that is power proportional (uses power proportional to its workload) and exposes slack (the ability to temporarily delay or degrade service to reduce power consumption). We describe a prototype cluster that consumes 60% less energy on typical workloads by being power proportional. Then, using month-long traces from a 9572 node computing cluster and a California wind farm, we show how a grid-aware scheduler can use workload slack to reduces dependence on non-renewable energy sources to 40% of its original level. Our results show that achieving the same wind penetration with energy storage alone would require sufficient battery capacity to run a cluster for five hours.

1 Introduction

Transitioning from fossil fuels to the use of sustainable energy sources is one of the most pressing challenges today. Use of variable renewable sources of electric power, *i.e.*, from the wind and the sun, is essential in addressing this challenge, because electricity constitutes 40% of overall energy consumption [36].

The fluctuating nature of these sources fundamentally limit their penetration in today's electric grid. Augmenting variable renewable supplies with backup generation and grid-scale storage has been proposed, but such backup generation can result in more greenhouse gas emissions than with no renewables at all [8] because backup plants must be left running, ready to makeup for drops in renewable supply, and effective grid-scale electricity storage solutions remain elusive [18]. We seek to substantially increase the renewable penetration limit by introducing *energy agile loads*: loads on the electric grid that provide mechanisms for controlling when they consume power or how much power they consume according to some policy.

Energy agile loads offer the opportunity to schedule demand when renewable power supplies are available. Designing the control algorithms and communication protocols required to match the demand of energy agile loads to the supply of renewable power sources is fundamentally a distributed systems scheduling problem, so we apply techniques commonly found in computer systems research to this broader challenge. As a concrete example and proof-of-concept, we design and evaluate a prototype energy agile computing cluster.

Computing clusters are a promising candidate for prototyping this cooperative grid management for several reasons. They are large loads, consuming 61 billion kilowatt-hours in the US in 2006 at a cost of about \$4.5 billion [37] and controlled by a small number of administrative domains. Moreover, they exhibit a fair amount of energy usage *slack*: the flexibility in the system to delay or adjust the consumption associated with the service it provides. Specifically, this slack comes from slightly delaying long running batch jobs and gracefully degrading interactive services.

In this paper, we implement a cluster using current hardware, running common batch and interactive workloads, that can adjust its power consumption. We then use simulation, driven by real-world computing workloads and wind farm traces, to explore several general policies for scheduling demand to enable using more intermittent renewable energy on the electric grid. Our results show that we can reduce dependence on traditional generation sources by over 60%. For a modestly-sized (one-thousand node) cluster, achieving the same with energy storage alone would require 5 MWh of storage or approximately 200 metric tons of lead-acid batteries (four shipping containers worth).

We believe similar results may be achieved for non-compute electric loads, as more such loads are connected to the Internet for communication and control. The techniques investigated here are relatively straightforward, are in use in other aspects of computing, and have natural analogs in other domains (*e.g.*, batch compute loads are analogous to appliances, interactive compute loads to lighting). A substantial portfolio of energy agile loads is necessary for feasibly attaining high-levels of renewable integration. Without energy agile loads, the maximum level of variable renewable penetration is severely limited, as discussed in the next section. Thus, the computer science techniques used here to design, implement, and evaluate an energy agile computing cluster have the potential for much broader impact. In this paper, we:

- Motivate energy agile loads as the most realistic method of transitioning the electric grid to sustainable energy sources.
- Formulate the problem of creating energy agile loads, presenting the key components: mechanisms and policies.
- Design, implement and evaluate a power proportional cluster running interactive and batch workloads with the mechanisms for shaping power consumption.
- Simulate and evaluate policies for controlling this cluster to maximize the use of renewable energy.
- Demonstrate an over 50% increase in the use of wind energy through intelligent scheduling. Achieving the same increase with storage alone would require 4 shipping containers worth of lead-acid batteries.

2 Background

The modern electricity grid delivers a perishable commodity from producers to consumers in real-time, over huge geographic distances with no intermediate storage or buffering. Since electricity generation was deregulated in the 1990s, various markets for trading electricity was developed. In bilateral trading, a single buyer and seller agree upon the price, quantity, and physical transfer of electricity. In competitive electricity pools, many buyers and sellers participate in a market operated by Independent System Operators (ISOs). ISOs provide a variety of functions, the most relevant being that of market maker. ISOs coordinate between three types of market participants: generators, transmission line operators, and consumers.

In most cases consumers place no orders and simply consume when they wish. System operators rely on statistical multiplexing of the individual demand of large numbers of these consumers to make the aggregate demand relatively predictable. Generators, under the direction of system operators, modulate their output on a variety of time-scales in response to changes in aggregate demand. In other words, loads are oblivious and non-dispatchable (uncontrollable), while supplies are reactive and dispatchable [4].

Most conventional generation resources such as oil, coal, natural gas, nuclear, and hydroelectric plants are known as dispatchable generation resources because they are controllable subject to restrictions on ramp rate. Slow-ramp plants (nuclear, coal) service the base load (minimum aggregate demand), with a portfolio of intermittent and fast-ramping peaker plants scheduled according to predicted variation. Increasingly, variable renewable sources such as wind and solar are coming onto the grid, driven partly by aggressive policy targets [9]. Renewables challenge the traditional grid paradigm because they are essentially oblivious and non-dispatchable supplies, governed by intermittent and variable factors like the wind and the sun, rather than a system operator.

The introduction of a large proportion of renewables into the current system can lead to serious grid instability unless considerable firming efforts are undertaken. Currently either natural gas-powered backup generation or grid-scale energy storage is used to supplement renewable supplies and reduce instability. Large-scale studies show that both have serious flaws: A California study found that using backup generation in a grid with predominantly renewable sources would generate more greenhouse emissions than one with no renewables [8], and a European study found that the amount of energy storage needed for full renewable penetration exceeded all of the potential storage capacity in Europe by more than an order of magnitude [18]. Published results of studies concerning the maximum level of wind penetration vary widely, but generally range from around 20% to 50% or more, depending on assumptions [8]. However, most studies to date have not seriously considered the impact of technological innovation on demand-side response, such as energy agile loads, on renewable power availability, other than for rare critical peak curtailment.

In a future electric grid with energy agile loads, one possible scenario is that such loads would interact with the ISOs, exchanging relevant information about consumption needs, electricity availability, price, and so on. For instance, energy agile loads could potentially participate in the real-time markets. How best to incorporate energy agile loads into existing the regulatory and economic frameworks in which electric grids operate is explored in work such as [6]. We later propose some reasonable scenarios for the purpose of our evaluation, but the regulatory and economic aspects of the problem are beyond the scope of this paper; we focus on the technical design aspects of designing such loads.

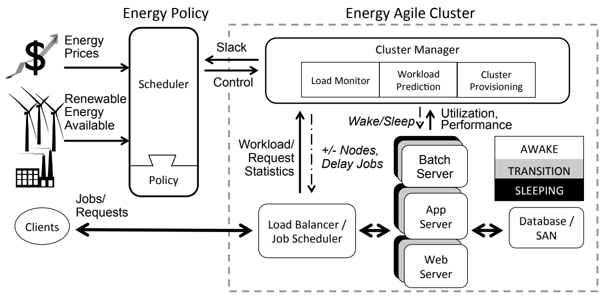


Figure 1: System architecture. An energy policy based on electricity prices and the available renewable energy governs the power consumption of a traditional three-tier, master-worker cluster.

3 Problem Formulation

We define an energy agile load as one whose demand profile, or power consumption over time, can be shaped without overly negative impact on the service it provides. We distinguish between mechanism: how a load scales or shifts its power consumption, and scheduling policy: when to scale or shift the power consumption. This notion of how much can be scaled and shifted is more concisely expressed as *slack*. Scheduling policies take advantage of the slack exposed through load-specific mechanisms.

3.1 Mechanism

For power consumption to be shaped by scheduling work, the mechanism must ensure that the load's power consumption closely tracks the rate of underlying work *i.e.*that the load is power proportional. A load which is efficient over its entire operating range is power proportional. However, many loads are inefficient at low levels of utilization. This inefficiency in scaling down is often an artifact of the engineering design process, which focuses on efficiency at high levels of utilization. Computers, especially high-performance servers are generally not power proportional loads. However, as shown in Section 5, it is possible to construct power proportional aggregate load out of non-power proportional components.

How a load scales or shifts its demand profile, (*i.e.*, the kind of slack it offers) is particular to that load. In computing, interactive loads can expose slack by reducing quality of service; for example, by reducing the completeness of search results or removing page elements. These techniques are already used for managing flash crowds, meeting SLAs, dealing with partial failures, and providing differentiated services. In batch computing, jobs often have deadlines; the temporal slack is correlated with how distant and how flexible the deadline is. The mechanism designer determines how much of the tradeoff between service degradation or delay and energy agility to expose, or it can be exposed by expressing the performance penalty as a function of slack.

3.2 Scheduling Policy

While the purpose of the mechanism is to expose slack, the purpose of the scheduling policy is to use this slack to match a load's demand profile to a supply profile. The scheduling policy must account for the nature of the slack. In the interactive case, slack is expressed as a level of degradation. In the batch case, slack is expressed as the time to a deadline. The policy must also account for the nature of the outside control signal that is available. This could be the amount of variable renewable power available (*e.g.*, in watts) or the real-time market price of electricity. If the price of renewable energy is set to be lower than that of less clean sources, an example policy could be "minimize the amount of money spent on power."

One subtle problem is that it is not clear what would happen if very large numbers of energy agile loads on the grid started responding to price or other signal in this way. It may, for instance, induce oscillations in the system. This suggests that some higher order controller, which isolates such effects or coordinates across loads, may be needed. Exploring this is ongoing and future work, as it requires first creating energy agile loads, the subject of this research.

There are practical limitations to evaluating load scheduling policies on the electrical grid, as to our knowledge, few, if any, large-scale grid testbeds exist and there are regulatory, political and economic questions about how renewables will be allowed to function on the grid. We use simulation with real traces from wind farms and historical California energy market prices to drive our policy evaluations. We make simple but justifiable assumptions about how renewables will be used and avoid getting into regulatory and political factors where there is often no clear right answer.

4 Energy Agile Computing Cluster

We designed and implemented a set of mechanisms and policies both an interactive web service cluster and a batch processing cluster. Figure 1 shows the architecture of our system.

4.1 Architecture

The cluster is based on the common cluster three-tier architecture: a tier of client-facing load balancing switches in front of a pool of servers, backed by shared storage [7]. we augment this design with a cluster manager. The cluster manager orchestrates the cluster components to provide slack. Sections 5.6 and 5.7 discuss both quality and temporal slack-providing mechanisms. There are two main challenges in designing such mechanisms. (1) In order to access the slack inherent in the computing workloads, the cluster manager must enforce power proportionality. (2) The mechanisms must provide a way to tradeoff between performance and energy agility. Section 5 details the design choices we made to retrofit power proportionality on a typical interactive web service and batch computing architecture, but at a high level, we capitalize on the fact that cluster services typically utilize a master-worker pattern, where the master is responsible for representing the ensemble, while distributing a request load over a collection of workers. We extend the master to manage the power state of the collection of workers. The worker platform is extended to support a mechanism to transition between active and sleeping. The cluster manager uses these internal capabilities to implement policy while interacting with an external signal.

The cluster manager also enforces energy policy to react to pricing or renewable availability signals; the design and evaluation of these policies is discussed in Section 6.

4.2 Methodology

The evaluation of our energy agile computing cluster design relies on both prototyping and simulation. We start with an evaluation our prototype energy agile load, and then modeling its interaction with large-scale wind farms and the broader electric grid using simulation. The prototype evaluation focuses specifically power consumption-shaping mechanisms. The simulation explores scheduling policies.

We drive both the prototype and the simulation using real-world traces; both for the load and the supply. For the interactive load, we use traces from Wikipedia, a real-world multi-tiered web application, so we implement an energy agile Wikipedia-clone using the same software used by the actual organization. For the batch load, we use traces from two real large-scale scientific computing clusters. The first is a month-long trace from a 112 node, 576 core, natural language processing and machine learning cluster used within our department. The second is a month-long trace from the $9,572$ node, $38,288$ core, Franklin cluster at the National Energy Research Scientific Computing Center (NERSC) [25]. Thus, we also implemented an energy agile version of Torque [32], a widely used, open source resource manager providing control over queues of batch jobs and distributed compute nodes.

The pricing signal we use is a trace of the real-time market price. It is determined by the electricity market and varies every five minutes based on the availability of renewable and traditional energy supplies. We use historic price data from the California real-time market, described in Section 2. Real-time market prices vary drastically, often ranging from \$-20/MWh[1] to over \$60/MWh

[1]At certain times energy consumers are paid to draw power due to unpredicted fluctuations in demand or intermittent supplies this can cause negative prices.

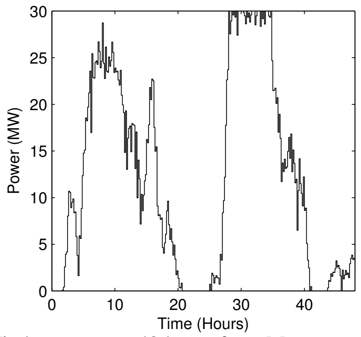


Figure 2: Wind power over 48 hours from Monterrey, CA wind farm. Wind power varies drastically, changing by up to 11MW in 20 minutes.

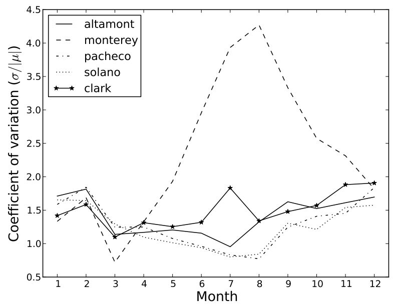


Figure 3: Wind variability over month long intervals at multiple wind farms. Variation is high not only over short time scales, but over longer time scales and across locations as well.

within a day as shown at the top of Figure 11.

We use wind speed and power data from the National Renewable Energy Laboratory (NREL) Wind Integration Dataset [24]. The dataset contains time series data in 10 minute intervals from wind farms located across the US. Each location is modeled to have 10 Vestas V90 3MW turbines and thus produces up to 30MW of power. We use the data from locations in California for our experiments. Figure 2 shows a 48 hour trace of wind power at a single location in Monterrey County, California. Wind power varies drastically, changing by up to 11MW in 20 minutes. Figure 3 shows the coefficient of variation – the standard deviation divided by the mean – for month-long intervals at different wind farms. The coefficient of variation changes significantly throughout the year and at different locations.

5 Cluster Mechanisms

Since the mechanisms must provide a way to scale and shift demand for power by scaling and shifting work, the first challenge to address is making sure that power consumption tracks work rate. The second challenge is to provide a way to actually scale and shift work with the appropriate tradeoff between agility and performance. These goals are accomplished by:

- Choosing efficient and agile hardware building blocks
- Closely monitoring work rate to maintain a margin of capacity overhead, called the overprovisioning factor
- Sizing the overprovisioning factor to maintain perfor-

mance guarantees while providing proportionality

- In the interactive case, using techniques for reducing the amount of work required to service a request to boost performance at a slight cost to quality
- For batch, using deadlines and run time monitoring

Note that each of these individual items are already commonly used in real computing systems for other purposes. We show how to combine them to provide energy agility. The following sections explain the importance of each of these points.

5.1 Overview

To study interactive workloads involving web requests, we implement a version of Wikipedia, a real-world multi-tiered web application. At its core, Wikipedia consists of a set of application servers running an open source PHP application called MediaWiki [29] in front of a set of MySQL back-end databases. Wikipedia makes ample use of caching, using front-end Squid reverse web cache servers, distributed memory cache `memcached`, PHP bytecode caches, and database caches. Wikipedia also uses several Linux Virtual Server (LVS) [41] servers for load-balancing.

Batch jobs are scheduled by Torque [32], a widely used, open source resource manager providing control over queues of batch jobs and distributed compute nodes. When submitting jobs to Torque, users specify the number of processors and amount of memory to be allocated, as well as the maximum running time. During job execution, the scheduler keeps track of the remaining running time of each job. We collect job execution traces using Torque's `showq` command to sample the cluster state at 1 minute intervals. We collected a one month trace of 128,914 jobs and extracted job start times, end times and user-specified maximum running times. Deadlines are defined as the start time plus the maximum running time.

The prototype runs on a 16-node cluster of dual-core Intel Atom boards (Section 5.2); The simplest way of achieving power proportionality would be to construct a compute cluster from power proportional components. Unfortunately, not even these relatively efficient Atom processors are power proportional; in general, server power draw varies only slightly with utilization. For example, operating at 30% utilization consumes at least 75% of peak power on most servers. Thus, adjusting utilization of the machines in a cluster does not provide sufficient control of the cluster's energy consumption. However, we can modulate the power consumption of the whole cluster by coarse grain control over the non-power proportional nodes.

Our system works by constantly monitoring incoming load and adjusting the system capacity in response by changing the number of machines kept in active and low-power standby states. We first add the ability to place the nodes into a sleep state (ACPI S3) when commanded over the network; Wake-on-Lan (WoL) is used to awaken a machine when necessary. Next, since a server should not be put to sleep when running jobs, we modify the load balancer/job scheduler to support placing jobs on a subset of the entire cluster. This way a machine can be shut down once all running jobs complete and it has been taken out of the active pool. Since, servers take time to transition from a standby or shutdown state to active (a challenge for serving low-latency interactive workloads) we maintain enough capacity to service the load and handle surges and spikes within the time it takes to bring up additional servers. We address this through rate prediction and provisioning algorithms.

5.2 Hardware

We surveyed a large number of platforms, paying particular attention to three characteristics: (1) operating range efficiency, (2) peak-to-idle power ratio, and (3) power management functionality, such as ACPI sleep states. To gauge operating range efficiency, we use an energy goodput metric of Joules per units of work at maximum throughput. The peak-to-idle power ratio directly relates to how efficiently it scales *down*; since many systems operate near peak power while performing little work. Finally, power management capabilities vary widely between different platforms.

We explored three classes of machines: server, mobile, and embedded Despite their wildly different performance numbers, the three different types of nodes exhibit comparable peak energy efficiency, consuming around 1J per response. Each node has an energy efficient operating range at which it attains both low energy per response and short response times. This range for the Atom 330 is shown in Figure 4. Although the Nehalem system provides both the best peak efficiency and the most overall capacity, its lack of fast ACPI sleep states means that a full shutdown is required to reduce the idle energy consumption to zero. Due to these limitations of the Nehalem and the low request-serving capacity of the BeagleBoards, we chose to build the cluster with Atoms. Specifically, we use the Atom 330 board which is inexpensive and was the only Atom system available at the time of construction. At the time of construction these systems were near best in energy efficiency. Since then, newer servers have been released with peak-to-idle ratios of up to 3, however, the idle power is still well over 100W [31].

5.3 Rate Prediction

The rate estimation component determines the amount of headroom needed to hide system ramp up latency and minimize the performance impact of power management. By accurately predicting the request rate we can bring up systems before they are needed and take them down

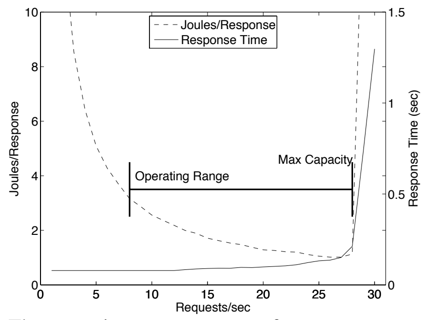


Figure 4: The operating range: range of request rates at which the system has high energy-efficiency and low response times.

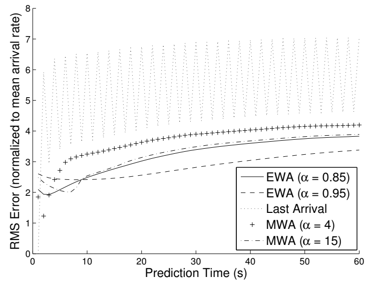


Figure 5: Comparison of different request rate prediction algorithms used to preemptively turn on servers, hiding the startup latency. Predictions should be accurate over typical 3-60 second startup times.

when not. We consider three types of rate predictors: last arrival, moving window average, and exponentially weighted average. Figure 5 shows that all averaging algorithms perform well on the wiki trace, with errors of less than 4% of the mean. Overall, the exponential weighted average with an α value of 0.95 is found to perform best for long prediction windows.

5.4 Capacity Provisioning

The critical factors are how much headroom needs to be reserved and how frequently the resources need to be provisioned. The cluster provisioning algorithm uses a bin packing algorithm to choose a set of machines, weighted by their efficiency, such that the sum of the system operating points is above the predicted load. This maximizes system utilization while preferring the most efficient systems.

We evaluate our provisioning algorithm on two metrics: energy efficiency and number of violations. The efficiency is a measure of how much energy the cluster uses compared to an oblivious policy; it is measured in Joules per request; lower numbers indicate more energy savings. Violations occur when a sudden burst of requests exceed the maximum capacity of the active cluster nodes. There is a trade-off between the aggressiveness of the power management and the number of violations that occur. We tune the aggressiveness of our provisioning algorithm by setting the operating point of each node to a fraction of the maximum capacity. This overprovisioning seeks to ensure that bursts can be absorbed without impacting performance. Whereas capacity planning on the time scale of months typically targets the neighborhood of twice the observed maximum hourly average, we find that at short time scales it is sufficient to maintain about 10% excess capacity.

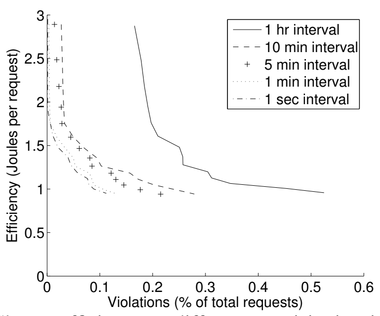


Figure 6: Cluster efficiency at different provisioning intervals. It is significantly less efficient to adjust to the load only once an hour versus every 10 minutes.

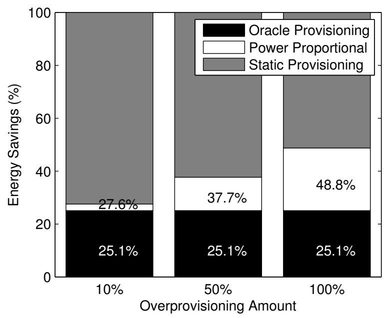


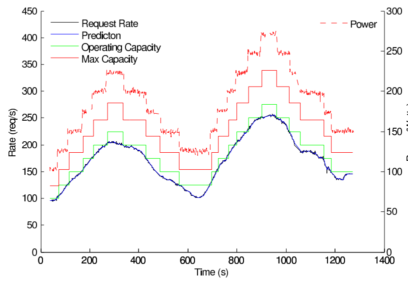
Figure 7: Overall energy comparison of static provisioning, our power proportional provisioning and an oracle provisioner against a one-week Wikipedia web request trace.

The provisioning algorithm is executed at regular intervals to reconfigure the cluster. Figure 6 explores the effects of running at different provisioning intervals at 10% headroom. We see that provisioning intervals under 10 minutes all perform comparably and much better than a once-an-hour provisioning rate. Overall, by using a higher provisioning rate we can achieve low violations with less overprovisioning than we would require at a lower rate.

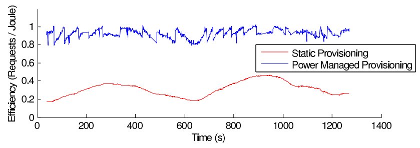
5.5 Power Proportionality

Taking the best combination of rate predictors and provisioning schemes, we compare the efficiency of our power proportional cluster to (1) a static provisioning scheme in which the capacity is sized using the standard provisioning rule of taking twice the maximum hourly load seen over the trace and (2) an oracle cluster manager which examines the full trace and chooses the minimum number of systems needed to handle the load.

Figure 8(a) shows the instantaneous request rate from the Wikipedia clients. Closely overlaid on this is the predicted rate 10 minutes ahead. The operating capacity tracks the request rate closely, with the steps determined by the node granularity. At all times, some headroom is



(a) Workload tracking



(b) Efficiency

Figure 8: Power proportionality for a sampled two day Wikipedia trace. The system accurately tracks the diurnal pattern of the trace with the power consumption varying commensurably with the request rate.

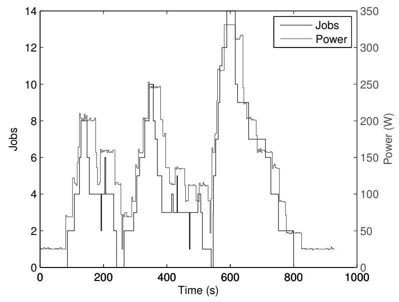


Figure 9: Workload and measured power consumption of a cluster running our power proportional Torque scheduler. Cluster power consumption varies with the workload. The system consumes 37% of the energy of a statically provisioned system.

maintained, but is tightest around 700 seconds in. The jittery graph above shows the measured power draw of the cluster: the variations are due to processor power state changes. Figure 8(b) normalizes the delivered performance by energy consumption. This is nearly constant at 0.9 requests per Joule, whereas standard Wikipedia on the same hardware has varying, lower efficiency.

Figure 7 summarizes the energy savings achievable at different levels of headroom for a simulated cluster of 250 nodes running the full Wikipedia workload. The power proportional cluster manager consistently saves over 50% of the energy consumed with a static provisioning scheme. Varying the overprovisioning amount, we trade-off between energy savings and performance. At 10% headroom, 0.15% of responses exceed our 300 ms response time requirement and we achieve over 70% power savings compared to static provisioning. This is within 10% of the optimal scheme.

Finally, we construct a power proportional batch system by instrumenting Torque to wake servers on demand and sleep idle machine after a timeout. Figure 9 shows the measured power consumption of our power proportional cluster running a scientific computing workload overlaid on the number of jobs. Our system consumes 37% of the power of a statically provisioned system.

5.6 Quality Slack

Although specific service degradations are application specific, there are a number of general techniques which can be used to reduce the amount of work required to service each request. They involve degrading service on one of several possible axes: freshness, latency, or quality. These techniques are closely related to research in *load shedding* for unloading a system when a particular resource is oversubscribed.

The crudest form of service degradation in the face of overloading is simply admission control, frequently performed in networks and either be explicit as in RSVP and SEDA or implicit as in RED. However, many interactive workloads have the ability to degrade service without merely refusing to service some requests. Modern web applications make tens to hundreds of backend web service calls when rendering a page or generating results. Slow or non-functioning service calls can be ignored to generate a slightly degraded result [1, 19, 23, 3, 12, 40]. Other options are to dynamically change which content is cached [20], return a random subsample of the results set [34], or remove low-value components of the result. Yet another option is for the service to intentionally delay the response to certain requests in order to cause clients to back off and lower their request rate. As a concrete example, in the case of an online shopping site, a degraded page containing cached versions of recommendations and less personalization, reviews or other dynamic content could significantly reduce the load; the system could also halt any expensive A-B tests in progress, and increase client cache durations.

In our prototype Wikipedia-clone, the wiki servers begin returning cached versions of pages instead of redrawing the page to reflect the last modifications, change time and new links to/from the page for each client request. Many of these changes could be nearly imperceptible to the user, and developing this type of flexibility will also benefit the site since the lower-resource degraded mode would also allow it to serve a large number of requests to better deal with flash crowds.

5.7 Temporal Slack

When load must be shed, jobs are delayed as permitted by available slack. An illustration of slack in batch jobs is provided in Figure 10. Job durations must be known ahead of time or predicted, as shown in prior work[13, 14, 27]. In practice, job length mispredictions are handled gracefully, the job simply completes past the deadline. We also assume jobs can be paused and resumed at 10 minute intervals. For parallel jobs we ensure that all tasks are scheduled to run concurrently.

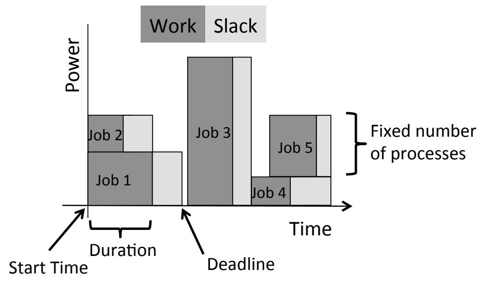


Figure 10: Schematic representation of batch jobs with slack. Each job arrives at a start time and runs for a fixed duration using a fixed number of nodes. The job must complete by the deadline.

Since Torque requires users to specify maximum running times, we use this number to infer an implicit deadline. Job deadlines are set to the start time plus the maximum running time. If the job duration is shorter than the maximum running time we assume the job can be postponed – this is the slack in the system. In a production energy agile system, users might explicitly specify deadlines.

6 Load Scheduling Policies

We evaluate two possible energy policies. Because the policy depends on the nature of slack that is exposed by the load, we must design the policy with the mechanism in mind. The first policy uses price as a signal to control an energy agile web services cluster. The policy is to minimize cost without increasing response latency or degrading the quality of responses by more than 50%. The second policy uses renewable energy availability as a signal to control an energy agile batch computing load. The policy is to maximize the use of variable renewable energy while avoiding missed deadlines. The four metrics used in the evaluation are:

- Total energy cost
- Non-renewable, grid-supplied energy used
- Renewable (wind) energy used
- Amount of renewable energy which was wasted (*i.e.*, generated but unable to be used)

While it might seem natural to design sophisticated prediction and scheduling algorithms to implement these policies, in the subsequent sections we show that simple algorithms perform quite well, approaching the theoretical maximum amount of wind energy use in the second scenario.

6.1 Grid-Stabilizing Interactive Web Services

The *policy* is to maintain a constant electricity cost while running the cluster. As real-time electricity prices vary with the availability of supply, this policy sheds load in proportion to the amount the current price exceeds the long-term average. This policy aims to match supply and demand on the electric grid by reducing price fluctuations. The *mechanism* is that cluster power consumption is reduced by degrading a fraction of incoming requests.

The policy returns the fraction of load that should be

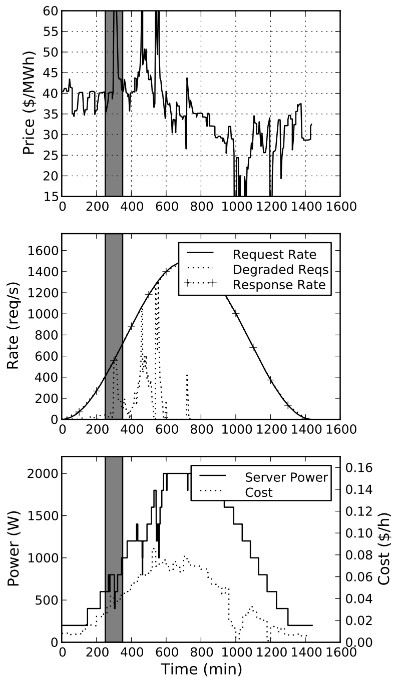
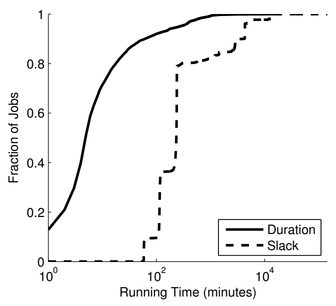


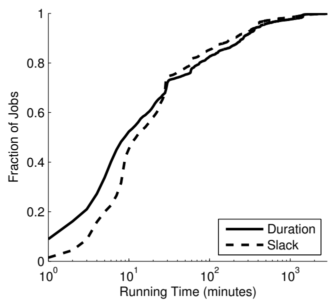
Figure 11: Load sculpting for a diurnal workload with a real energy price control signal. In the highlighted band, the load sculpting algorithm reacts to a price spike by degrading a fraction of requests which reduced the cluster power consumption by 50%.

shed to maintain constant cost. In response, the cluster manager instructs the load balancer to mark a fraction of all incoming requests as degraded. This fraction is determined based on the amount of energy required to generate a degraded response compared to a normal response. Specifically, $f_d = (1-\hat{L})/(1-E_d)$ where f_d is the fraction of requests to be degraded, $\hat{L}$ is the desired load as a fraction of old load and E_d is the work required to service a degraded request as a fraction of the work required for a normal request. Degraded requests are marked with a custom HTTP header which is recognized by the web server and generates a simplified response

Figure 11 shows the response of a simulated cluster to pricing on the California ISO spot market against a typical diurnal load profile. The top graph shows real-time energy prices over a day. These are highly variable, spiking when energy is scarce, requiring that more expensive plants be used, and dropping, occasionally below zero, when excess energy is available due to unexpected supply or unpredicted drops in demand. The middle graph shows the portion of the requests that are degraded. We highlight in gray the system's reaction to the first large price spike. The cluster manager responds by degrading some requests but maintaining the response rate. Here again, all requests are serviced. The overall cluster power consumption drops by 50% and the the cost of running the cluster is reduced by 24%. The daily energy costs are reduced by 6%, but more importantly, grid stability is improved by shedding up to 50% of the load during peak periods.



(a) PSI scientific computing cluster. Average job duration 55 min, slack 68 min.



(b) NERSC Franklin cluster. Average duration 98 min, slack 68 min.

Figure 12: CDF of job duration and slack for 112 node department cluster and 9,572 node NERSC cluster.

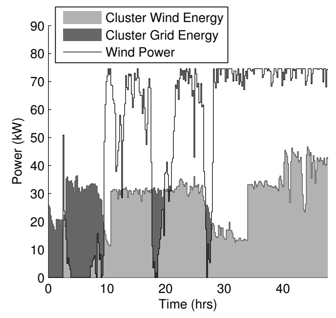
6.2 Low-Carbon Batch Scientific Computing

The *policy* is to maximize use of renewable energy by using slack in the workload to delay jobs when renewable energy is scarce while ensuring that job deadlines are met. The *mechanism* is to delay jobs as permitted to shed load.

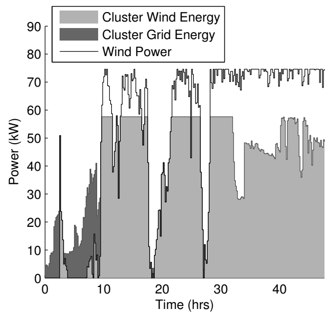
Figure 12 shows CDFs of job durations and slack for these workloads. The average job duration for the PSI cluster was 55 minutes and the average slack was 17 hours. Many jobs have 24 hour deadlines resulting in a high average slack. The NERSC cluster has more evenly distributed job durations and slack. The average duration is 98 minutes and the average slack is 68 minutes.

We begin by establishing the baseline performance of grid-oblivious scheduling. This scheduler is equivalent to the schedule of jobs on the original Torque cluster. It never postpones jobs, running each task as resources become available. However, we do assume the cluster is power proportional, putting servers to sleep when it is not fully utilized.

Figure 13(a) shows a 48 hour slice of the cluster power consumption when running the PSI workload with run-immediately scheduling. Overlaid on top is a plot of the power output from the Monterey wind farm scaled such that the average power of the wind trace is equal to the average power consumed by the cluster running the workload. This results in a wind trace whose peak



(a) Run-immediately, oblivious scheduling



(b) Greedy, grid-aware scheduling

Figure 13: Wind supply and cluster load overlaid. The grid-oblivious schedule obtains 54% of its energy from the grid. The greedy, grid-aware, schedule uses only 30% grid and 70% wind energy.

energy generation capacity is approximately 125% of the peak energy usage of the cluster. Wind power is shown in black, wind energy that is used by the cluster is shaded light gray and energy from the grid is shaded dark gray. The cluster power consumption is typically around 30kW. When wind output is high the cluster obtains all of its energy from the wind, but during drops in output the cluster relies entirely on outside, grid energy. Over the whole trace only 46% of cluster energy comes from wind, the remaining 54% is obtained from the grid. 54% of total wind energy is wasted.

We construct a simple grid-aware scheduler that uses slack in job scheduling to maximize the use of wind en-

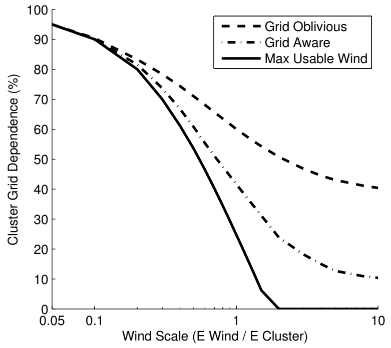


Figure 14: Fraction of cluster energy coming from wind for varying sizes of wind farm. The grid-aware schedule always uses at least 75% of the maximum usable wind energy. It typically performs 45% better than grid-oblivious scheduling.

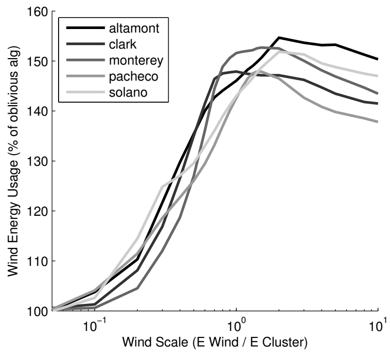


Figure 15: Improvement of grid-aware over oblivious scheduling for different wind traces.

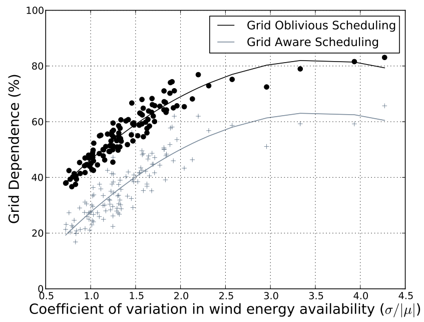


Figure 16: Performance of scheduling for increasing wind power variability. As coefficient of variation in the wind power increases, the amount of wind energy used decreases. Gird-aware scheduling consistently requires less grid energy than oblivious scheduling.

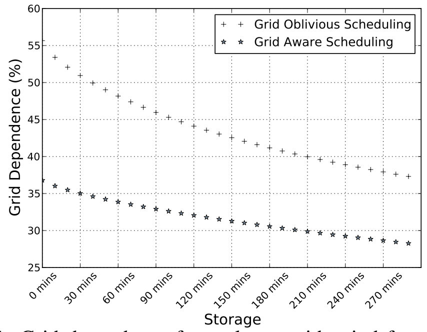


Figure 17: Grid dependence for a cluster with wind farm and varying amounts of battery storage. Using grid-oblivious scheduling, five hours of storage is required to match the performance of the grid-aware algorithm with no storage.

ergy and minimize grid demand. The scheduler uses an online algorithm, scheduling jobs for the next 10 minutes in each interval. For each time interval, the list of incomplete jobs is sorted by slack. A power budget is set to the current available wind power. Jobs that must run to complete by the deadline, *i.e.*, jobs with zero slack, are scheduled first, irrespective of the available wind. If the power budget is not exceeded then jobs with non-zero slack are scheduled up to the power budget.

Figure 13(b) shows a 48 hour slice of the grid-aware schedule. The scheduler runs fewer jobs in the "valleys" when wind output drops. It also utilizes the whole cluster, using approximately 60 kW, when wind output is high. Overall the grid-aware schedule uses 30% grid energy and 70% wind energy; 30% of wind energy is wasted.

We explore three key parameters that impact the performance of the grid-aware scheduling: wind scale, wind variability and workload slack. First we consider the size of the wind farm and its peak output. Naturally, a larger wind installation is able to supply a larger fraction of cluster energy consumption. We model different available wind sources by scaling wind traces by a constant factor chosen to yield a desired ratio of total wind energy to total cluster energy. For example, at a scale of 1 the total wind energy over the duration of the workload is equal to the cluster energy need, as in Figure 13.

Figure 14 shows the fraction of cluster energy obtained from the grid for different wind scales when running the PSI workload with the Monterey wind farm trace. A smaller grid dependence is better, since it results in more wind energy use. The maximum usable wind energy is computed by integrating the wind power trace below the maximum power of the cluster. Since the cluster is of finite size, it cannot consume above a certain power limit. The maximum usable energy is a loose upper-bound on the performance of the scheduler since it does not consider the workload characteristics. Grid-aware scheduling is always within 75% of this bound and generally uses 45% more of the wind energy than grid-oblivious scheduling. Figure 15 shows improvement of grid-aware over oblivious scheduling for different wind traces and wind scales. In all cases, as the availability of wind increases, however, the grid-aware scheduler quickly outpaces the grid-oblivious scheduler.

Of critical concern is the effect of wind variability on the effectiveness of energy agility. Figure 16 shows grid dependence versus the coefficient of variation of the wind trace. We observe that the amount of grid energy required increases as the wind power becomes more variable. However, grid-aware scheduling uses consistently more wind energy for all levels of variability. Moreover, the improvement in wind use over the grid-oblivious scheduler is more pronounced at larger variability.

We also consider a hybrid design with both a wind farm and battery storage attached to the cluster. Energy storage is commonly used to smooth out intermittent wind power output, although it remains a technological challenge to do so cost-effectively and the environmental impact of the batteries is of concern. Figure 17 shows the percent of grid energy required versus the battery storage capacity measured in minutes of running the cluster. The results are averaged over all months of all wind traces. Using storage improves wind energy use for both grid-oblivious and aware algorithms. But over 5 hours of storage capacity is required for the grid-

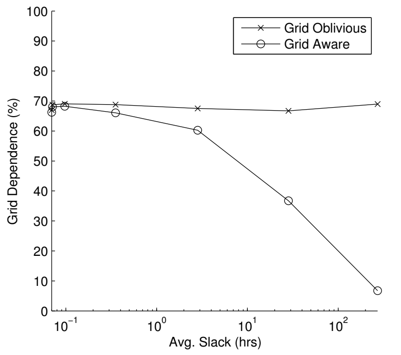


Figure 18: Grid dependence for varying amounts of slack.

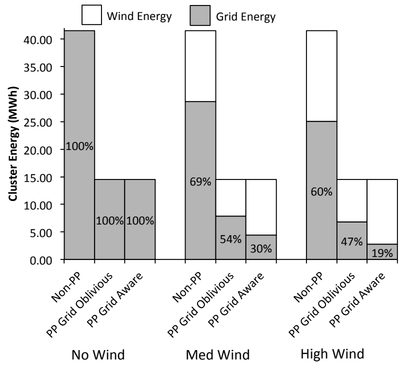


Figure 19: Cluster energy consumption and fraction of wind energy used. Power proportionality (PP) significantly reduces energy consumption. Grid-aware scheduling reduces grid dependence by between 44% and 60% and increases wind energy use by over 50%.

oblivious system to match the performance of the grid-aware system without storage. For our NERSC cluster this would mean 8MWh of storage. Using lead acid batteries this would weigh approximately 200 metric tons and occupy 4 shipping containers.

Figure 18 explores the impact of workload slack on system performance. We use a wind trace scaled to match the energy needs of the cluster. For the workloads, we generated traces parameterized by five characteristics: average job interarrival rate, average job duration, average job slack, average number of concurrent jobs, and average cluster utilization. All parameters but slack are held constant, with slack varying by powers of 10. As slack increases, the grid dependence drops drastically when using the grid-aware scheduler. Given enough slack the system can make use of all usable wind energy.

Finally, Figure 19 summarizes the energy consumption and renewable energy use of the cluster for different amounts of wind. "Med Wind" is scaled to have the same amount of energy as used by the power proportional cluster. "High Wind" is scaled to have two times the energy of the power proportional cluster. In all cases, power proportionality, labeled PP, significantly reduces the energy consumption of the cluster. For a fixed wind trace, reducing consumption increases the fraction of energy coming from wind because there are fewer times when load exceeds available wind energy. Beyond power proportionality, our grid-aware scheduling further reduces grid dependence by 44% and 60%, depending on wind scale. Overall, we reduce grid energy consumption from 42 MWh used by a typical non-power proportional cluster without a wind farm to 2.8 MWh for a power proportional system with gird-aware scheduling and wind.

7 Related Work

In this paper we described how a traditional computing cluster can be made into a load that is energy agile. Such loads offer mechanisms for scheduling their power consumption and thus can be made to use renewable energy. Energy agile loads must first be power proportional [5], so that scheduling work will effect cluster power. Then the load can be used to explore scheduling policies for using renewable energy. Both power proportionality and powering IT loads with renewables have been the subject of much recent research. Below we describe some of this work to highlight where our work differs.

7.1 Power Proportionality

Single node solutions: Many early solutions [17, 22, 39] attempted to reduce platform energy usage by scaling down the CPU using techniques such as Dynamic Voltage/Frequency Scaling (DVFS). However, the CPU is only one of many components in the computer that consumes energy.

Distributed system solutions: [10, 26] apply dynamic provisioning strategies for web clusters, sometimes trading off user experience for energy usage [11]. Rabbit [2] and Sierra [35] are examples of distributed file systems that reduce the number of powered-up servers while still providing guarantees on the number of data replicas available. [38] modifies the Hadoop framework and HDFS to enable applications written using the framework to be power-aware. All of these techniques are applicable to developing energy agile systems, since a proportional system is the substrate for an energy agile system.

7.2 Powering Data Centers with Renewable Energy

Multiple data centers: Geographical load balancing across multiple data centers can be used to reduce electricity bills by exploiting regional differences in electricity prices [28] or to increase the amount of renewable energy used [33, 21]. Our work focuses on improving renewable energy use within a single data center.

Power budgets: Power capping, blinking, and duty-cycling servers between sleep and running states, have been proposed as ways of imposing cluster power budgets that could be set to the amount of currently available renewable energy [30, 15]. Our approach differs from these efforts because we consider both the slack available in the workload and the currently available renewable energy to construct a schedule ensuring that deadlines are not missed.

Power scheduling: Similar to this work, [16] schedules batch workloads to maximize the use of renewables. Our work differs in considering a mix of batch and interactive workloads running on a cluster using more variable and less predictable wind instead of solar energy sources. We also show results from a much larger, over $9,000$ node cluster at high utilization.

8 Conclusion

We propose extensions to cluster hardware and software which enable cluster management tools to control the power state of managed machines to effect energy related goals. Much of the rest of our study is unconventional by several measures. Instead of justifying architectural modifications on the basis of raw performance improvements or even cost savings, we instead explore a service that data centers are well-equipped to provide: modulating power and scheduling work around a control signal to improve the operation of the rest of the grid. In particular, we demonstrate a reduction in dependence on legacy grid generation by 50%, even when using the most difficult and unpredictable renewable resource, and an equivalence between having an agile cluster and five hours of energy storage. Considering that an agile cluster can be implemented only in software and potentially scales well, while five hours of storage costs millions of dollars for a modestly-sized data center, we feel that this approach is promising.

References

[1] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center TCP (DCTCP). *SIGCOMM* (2010).

[2] AMUR, H., CIPAR, J., GUPTA, V., GANGER, G. R., KOZUCH, M. A., AND SCHWAN, K. Robust and flexible power-proportional storage. In *SoCC* (2010).

[3] ARON, M., DRUSCHEL, P., AND ZWAENEPOEL, W. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *SIGMETRICS* (2000).

[4] BARNETT, T., PIERCE, D., STEINNEMANN, A., ALTALO, M. G., DAVIS, T. D., GREENING, L., HALE, M., AND GAUSHEL, D. Improving weather and load forecasting for the california independent system operator.

[5] BARROSO, L. A., AND HÖLZLE, U. The case for energy-proportional computing. *Computer 40*, 12 (2007).

[6] BITAR, E. Y., RAJAGOPAL, R., KHARGONEKAR, P. P., POOLLA, K., AND VARAIYA, P. Bringing wind energy to market. *submitted to IEEE Transactions on Power Systems* (2010).

[7] BREWER, E. A. Lessons from giant-scale services. *IEEE Internet Computing* (2001).

[8] CALIFORNIA COUNCIL ON SCIENCE AND TECHNOLOGY. California's energy future – the view to 2050.

[9] CALIFORNIA ENERGY COMMISSION. California renewable energy overview and programs.

[10] CHASE, J., ANDERSON, D., THAKAR, P., VAHDAT, A., AND DOYLE, R. Managing energy and server resources in hosting centres. In *SOSP* (2001).

[11] CHEN, G., HE, W., LIU, J., NATH, S., RIGAS, L., XIAO, L., AND ZHAO, F. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI* (2008).

[12] FOX, A., AND BREWER, E. A. Reducing WWW latency and bandwidth requirements by real-time distillation. In *WWW* (1996).

[13] GANAPATHI, A., KUNO, H., DAYAL, U., WIENER, J. L., FOX, O., AND JORDAN, M. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE* (2009).

[14] GANAPATHI, A. S., CHEN, Y., FOX, A., KATZ, R. H., AND PATTERSON, D. A. Statistics-driven workload modeling for the cloud. In *SMDB* (2010).

[15] GMACH, D., ROLIA, J., BASH, C., CHEN, Y., CHRISTIAN, T., SHAH, A., SHARMA, R., AND WANG, Z. Capacity planning and power management to exploit sustainable energy. In *CNSM* (2010).

[16] GOIRI, I., LE, K., HAQUE, M. E., BEAUCHEA, R., NGUYEN, T., GUITART, J., TORRES, J., AND BIANCHINI, R. Greenslot: Scheduling energy consumption in green datacenters. In *SC11* (2011).

[17] GRUNWALD, D., LEVIS, P., FARKAS, K. I., III, C. B. M., AND NEUFELD, M. Policies for dynamic clock scheduling. In *OSDI* (2000).

[18] HEIDE, D., GREINER, M., VON BREMEN, L., AND HOFFMANN, C. Reduced storage and balancing needs in a fully renewable european power system with excess wind and solar power generation. *Renewable Energy* (2011).

[19] HOELZLE, U., AND BARROSO, L. A. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[20] IHM, S., PARK, K., AND PAI, V. S. Wide-area network acceleration for the developing world. In *USENIX ATC* (2010).

[21] LIU, Z., LIN, M., WIERMAN, A., LOW, S. H., AND ANDREW, L. L. Greening geographical load balancing. In *SIGMETRICS* (2011).

[22] LORCH, J. R., AND SMITH, A. J. Improving dynamic voltage scaling algorithms with pace. In *SIGMETRICS* (2001).

[23] MELNIK, S., GUBAREV, A., LONG, J. J., ROMER, G., SHIVAKOMAR, S., TOLTON, M., AND VASSILAKIS, T. Dremel: Interactive analysis of web-scale datasets. In *VLDB* (September 2010), vol. 3.

[24] NATIONAL RENEWABLE ENERGY LABORATORY. Wind integration datasets.

[25] National energy research scientific computing center. `http://www.nersc.gov`.

[26] PINHEIRO, E., BIANCHINI, R., CARRERA, E. V., AND HEATH, T. Dynamic cluster reconfiguration for power and performance.

[27] POLO, J., CARRERA, D., BECERRA, Y., TORRES, J., AYGUAD, E., STEINDER, M., AND WHALLEY., I. Performance-driven task co-scheduling for mapreduce environments. In *NOMS2010* (2010).

[28] QURESHI, A., WEBER, R., BALAKRISHNAN, H., GUTTAG, J., AND MAGGS, B. Cutting the electric bill for internet-scale systems. In *SIGCOMM* (2009).

[29] RAHMAN, M. *MediaWiki Administrators' Tutorial Guide*. Packt Publishing, 2007.

[30] SHARMA, N., BARKER, S., IRWIN, D., AND SHENOY, P. Blink: Managing server clusters on intermittent power. In *ASPLOS* (2011).

[31] SPEC CORP. SPECpower ssj2008, 2011.

[32] STAPLES, G. Torque resource manager. In *SC* (2006).

[33] STEWART, C., AND SHEN, K. Some joules are more precious than others: Managing renewable energy in the datacenter. In *HotPower* (2009).

[34] TATBUL, N., ÇETINTEMEL, U., ZDONIK, S., CHERNIACK, M., AND STONEBRAKER, M. Load Shedding in a Data Stream Manager. In *VLDB'03* (2003).

[35] THERESKA, E., DONNELLY, A., AND NARAYANAN, D. Sierra: Practical Power-proportionality for Data Center Storage. In *EuroSys* (2011).

[36] U.S. ENERGY INFORMATION ADMINISTRATION. Primary energy consumption by source and sector.

[37] U.S. EPA. Epa report on server and data center energy efficiency. *ENERGY STAR Program* (2007).

[38] VASIĆ, N., BARISITS, M., SALZGEBER, V., AND KOSTIC, D. Making cluster applications energy-aware. In *ACDC* (2009).

[39] WEISER, M., WELCH, B. B., DEMERS, A. J., AND SHENKER, S. Scheduling for reduced cpu energy. In *OSDI* (1994).

[40] WELSH, M., CULLER, D., AND BREWER, E. Seda: an architecture for well-conditioned, scalable internet services. In *SOSP'01* (2001).

[41] ZHANG, W. Linux virtual server for scalable network services. *Ottawa Linux Symposium* (2000).