GreenCassandra: Using Renewable Energy in Distributed Structured Storage Systems

William Katsak[‡], Íñigo Goiri[†*], Ricardo Bianchini[†‡], Thu D. Nguyen[‡]
[‡]Rutgers University [†]Microsoft Research
Email: {wkatsak, ricardob, tdnguyen}@cs.rutgers.edu Email: {inigog, ricardob}@microsoft.com
Technical Report DCS-TR-714, Dept. of Computer Science, Rutgers University, April 2015

***Abstract*—On-site generation of renewable ("green") energy can help to significantly reduce grid ("brown") energy consumption, and correspondingly the carbon footprint, of datacenters. However, it is challenging to use green energy generated from sources such as solar and wind because energy production is variable. In this paper, we investigate how to manage an interactive service, where response time is a critical performance metric, to maximize the benefits of green energy produced from these sources. Specifically, we design, prototype, and evaluate a distributed structured storage system, GreenCassandra, which is representative of a class of important subsystems underlying many interactive cloud services. Our proposed approach predicts the production of solar energy, and then controls the number of active nodes to manage energy consumption while respecting a response time SLA. When green energy is available, GreenCassandra may activate extra servers to build up slack with respect to the SLA. When using brown energy, GreenCassandra deactivates servers to reduce energy consumption, leveraging any built-up performance slack, while observing constraints imposed by the SLA. Evaluations show that GreenCassandra can use a heuristic green-energy-aware policy to decrease brown energy consumption and cost by up to 28% and 29%, respectively. Further, these savings are very close to those achievable by an optimization-based policy that has perfect knowledge of future workload and green energy production.**

I. Introduction

Datacenters consume an enormous amount of electricity, and this consumption is growing rapidly. Estimates for 2010 indicate that datacenters consume 1.5% and 2% of the total electricity used world-wide and in the US, respectively [1]. Further, this consumption likely grew by over 60% between 2011 and 2012 [2]. This electricity consumption is both expensive and leads to high carbon emissions, since most of the electricity is produced by burning fossil fuels.

As a result, there is rising interest in "green" datacenters powered at least partially with on-site generation (or "self-generation") of renewable ("green") energy from sources such as wind and solar. Self-generation is attractive for multiple reasons, including (1) the fact that power transformations and long-distance transmission may incur energy losses of up to 15% [3]; (2) the ability to survive grid outages, which are common in some developing countries; (3) reduced carbon emission (along with the marketing and/or public relations benefits of being "green"), and (4) eventual lowering of costs (for example, Goiri *et al.* argue that the installed capital cost for the solar energy system of a green micro-datacenter can be recovered from electricity cost savings in less than 10 years [4]).

A number of companies have built renewable power plants co-located with their datacenters. For example, Apple has built two 20MW solar arrays next to their datacenter in Maiden, NC [5]. Green House Data [6], AISO [7], and GreenQloud [8] are cloud providers that operate green datacenters mostly (or entirely) powered by wind and/or solar energy.

A challenge for maximizing the benefits of green energy generated from sources such as solar and wind is that their production is variable. For example, photo voltaic (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 (net metering). However, this approach has significant disadvantages: (1) batteries are expensive[1] and common types of batteries (e.g., lead-acid) are bad for the environment, (2) batteries incur energy losses, (3) net metering incurs losses and is not always available, 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 the management of cluster-based interactive services to better match energy demand to green energy production. A key driver of our study is the fact that response time is a critical performance metric so that incoming load cannot be deferred. The specific system that we study is a distributed structured storage system, which is representative of a class of important subsystems underlying many interactive cloud services.

In particular, we propose and evaluate GreenCassandra, a distributed structured storage system that seeks to maximize green energy consumption and minimize brown energy cost, *while preserving a response time service-level agreement (SLA)*. GreenCassandra achieves its objectives by predicting the production of green energy in the near future (up to 48 hours ahead), and dynamically adjusting the number of active servers. It generally attempts to deactivate servers not necessary for achieving its data availability and performance objectives under an offered load to conserve energy. However, when it expects excess green energy, it will activate additional servers to build up "slack" with respect to the SLA. For example, it might activate enough servers to service 99.9% of incoming client requests within a target response time when the SLA only requires 99%. Then, when GreenCassandra has to use brown energy, it expends any accumulated slack to further reduce brown energy consumption. For example, for a short period of time, it might deactivate servers so that only 98% of the incoming client requests are serviced within the target response time. In this way, GreenCassandra attempts to *temporally shift energy consumption despite servicing a non-deferable workload*.

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

[1]In fact, Goiri et al. found that the cost of batteries is currently not amortizable when batteries are actively used as a power source in a green datacenter unless the workload is deferrable [4].

We have designed four energy- and green-energy aware scheduling policies for GreenCassandra, targeting datacenters partially powered by solar energy. Each policy assumes a different amount of knowledge about the system itself, future green energy production, and future load. We have implemented these policies in a prototype GreenCassandra system, and evaluated our implementation in Parasol, a solar-powered micro-datacenter at Rutgers [4]. Our evaluation uses three workloads, each constructed to follow a trace of a real system (Ask.com, MS Hotmail, and MS Messenger).

For repeatability and proper scaling, our evaluation uses scaled-down traces of solar energy production from Parasol. Evaluation results show that GreenCassandra can reduce brown energy consumption and cost by up to 28% and 29%, respectively, compared to a baseline policy that is energy-aware but not green-energy-aware, when observing an SLA requiring the 99^{th} percentile response time for requests within an accounting period (1 day) to be no more than a target value (75ms). These results show that GreenCassandra can indeed leverage variable green energy production to significantly reduce brown energy consumption and cost.

In summary, we make the following contributions: (1) we introduce GreenCassandra, a distributed structured storage system designed for green datacenters partly powered by on-site solar energy generation; (2) we introduce scheduling policies that use information about electricity prices and green energy production to reduce electricity cost; (3) we demonstrate that energy consumption can be shifted temporally to better match variable green energy production even when the load must be serviced immediately (i.e., the load is not deferable); and (4) we present extensive evaluation results for GreenCassandra, isolating the impact of the different amounts of knowledge in the policies, and exploring sensitivity to various parameters.

II. Related Work

Lo *et al.* [9] have characterized two on-line Google workloads to show that the underlying clusters frequently operate at low and medium utilization. Then, they show that PEGASUS, a controller that dynamically adjusts CPU energy consumption, can improve cluster power proportionality while maintaining a response time SLA. Our baseline policies are similar to PEGASUS, and represent the state of the art energy-aware interactive services. However, Lo *et al.* do not consider green energy, and they only control the energy consumption of CPUs whereas we deactivate entire servers, which raises data availability concerns but can lead to higher energy savings.

Krioukov *et al.* [10] have also studied energy-agile interactive services, where the service power demand can be adjusted based on electricity pricing signals to minimize cost. They do not predict green energy production and do not consider building up performance slack as GreenCassandra does. They also study a simpler read-only service and a simpler SLA.

Previous works have sought to improve the power-proportionality of storage systems [11]–[13]. The data layout of GreenCassandra is inspired by [11]. Amur *et al.* [12] did not study energy management policies, while Pinheiro *et al.* [11] and Thereska *et al.* [13] did not consider green energy. Further, they did not target maintaining a response time SLA.

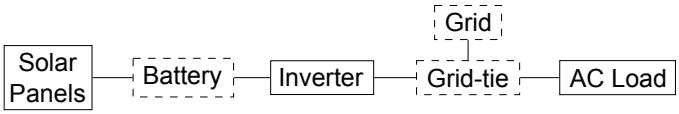


Figure 1. Components of a (partially) solar-powered computer system. Dashed boxes represent optional components.

Sharma *et al.* [14] have proposed BlinkFS, a distributed storage system that can adjust its power demand to match intermittent power production. BlinkFS can arbitrarily lower system power draw, but its design can lead to intermittent data availability and high response time when operating in low power states. In contrast, GreenCassandra is constrained to maintain constant data availability to preserve the performance SLA even in low power states.

A number of works have explored exploiting green energy or electricity pricing variation because of variable green energy availability for scheduling batch workloads [4], [10], [15], [16]. However, the main mechanism used to shape power demand is temporal delay of jobs, which is not appropriate for the interactive services that we consider. Aksanli *et al.* [17] considered green-energy aware scheduling of a mixed batch and interactive workload. However, the green energy scheduling only targeted the batch jobs.

Finally, a number of works have considered increasing use of green energy, reducing brown energy/power consumption, and/or reducing electricity costs in the context of load distribution across multiple datacenters [18]–[24]. In general, the approaches for load distribution across multiple datacenters are different than our for managing the energy consumption of a storage cluster within a single datacenter. Le et al. [20] did consider a response time SLA similar to our work. Similar to some of our policies, one of their policy can also trade periods where short-term performance can degrade beyond the SLA for lower energy consumption/cost.

III. Background

A. Green Datacenters

Solar and wind are two of the most promising green energy technologies, as they do not cause the environmental disruption of hydroelectric energy and do not have the waste storage problem of nuclear energy. In this paper, we assume that the datacenter generates its own PV solar energy. (Except for our solar energy predictions, our work is directly applicable to wind energy as well.) There are multiple ways to connect solar panels to a datacenter. Figure 1 shows a general setup. The solar panels can be connected to batteries for storing excess energy during periods of sunlight and discharging it during other periods. The DC electricity produced by the panels must be converted to AC electricity using an inverter to drive the IT and cooling equipment. The datacenter must also be connected to the electrical grid via a grid-tie device if it must be operational even when solar energy is not available. Where net metering is available, it is possible to feed excess solar energy into the grid for a reduction in brown energy costs.

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

B. Cassandra

Cassandra [25] is a popular distributed structured storage system that supports a table abstraction similar to that of BigTable [26]. Each table contains rows, columns, and column families. Data items are accessed by addressing row keys, and possibly column family and column names.

Cassandra is implemented as a key/value storage system using a distributed hash table (DHT) based on Amazon's Dynamo [27]. Each set of cells defined by a (row key, column family) pair is a data item. The row key is used to hash each data item into a 127-bit keyspace. Each server in a Cassandra cluster is assigned a token within the keyspace, and the servers are logically arranged in a ring in increasing order of tokens. Each server stores the data items in the portion of the keyspace between the server's token and the token of its predecessor in the ring. Each server maintains a table of information for all peers in the cluster for routing access requests. This table is kept loosely consistent using a gossiping protocol.

When configured to store N replicas of each data item, Cassandra's default data placement strategy is to place the first replica of a data item on the server responsible for the part of keyspace that the data item hashes into. The remaining $N-1$ replicas are placed on the next $N-1$ servers in the ring. In this manner, the replicas can always be located by hashing the row key and walking the ring. We call each server that stores a replica of a data item an *owning server* of that data item.

Read/write requests can arrive to any server in the cluster. A server receiving a read or write request is responsible for coordinating the servicing of that request, and so is called the *coordinator*. When servicing a write, the coordinator records the write in the local disk as a *hint* if one or more owning servers are known to be offline (because of failures).

Consistency model. Cassandra uses a tunable consistency model based on the number of replicas that must be read/written for a request to successfully complete. Writes are serialized using a user-provided timestamp. Each replica of a data item stores the timestamp of the last write.

Assuming there are N replicas of each data item, the main consistency policies and the coordinator's actions for reads are:

One: query any one of the owning servers.
Quorum: query $\lceil (N+1)/2 \rceil$ owning servers.
All: query all N owning servers.

Once the coordinator has received replies from all queried owning servers, it completes the read with the most recent version found.

To service a write, the coordinator forwards the write to all N owning servers. Then, the main consistency policies and the coordinator's subsequent actions are:

Any: wait for any forwarded write or the local write of a hint to complete.
One: wait for at least one forwarded write to complete.
Quorum: wait for at least $\lceil (N+1)/2 \rceil$ forwarded writes to complete.
All: wait for all N forwarded writes to complete.

The coordinator completes the write once it has received the requisite number of replies.

It is important to keep in mind that each write is sent to all owning servers, regardless of the specified consistency policy. The specified policy then determines how many owning servers (if any) must successfully perform the write before the coordinator can reply with a success to the client.

Background compaction. Cassandra stores data on disk using immutable files called SSTables similar to BigTable [26]. Writes are first stored in an in-memory buffer. When the buffer fills up, it is written to an SSTable, which is structured for fast lookup of data items. As time passes, Cassandra will accumulate multiple SSTables with data that has been overwritten. Thus, Cassandra needs to periodically compact these SSTables to reclaim storage space as well as prevent degradation of read performance.

IV. Structured Storage in Green Datacenters

We propose GreenCassandra, a distributed structured storage system for datacenters powered by PV solar panels and the grid. As already mentioned, GreenCassandra implements energy-management policies that leverage green energy and adjust the number of active servers to reduce brown energy consumption and cost, while observing a performance SLA. In this section, we first describe GreenCassandra's power malleable architecture, and then the energy- and green-energy-aware policies that we have designed for GreenCassandra.

A. Power Malleability

The mechanisms that we use to make GreenCassandra power malleable have mostly been proposed/used before (e.g., [11], [13], [28]). However, we have had to adapt them, especially to account for Cassandra's consistency model.

Data availability. We partition servers in a GreenCassandra system into a *covering* subset [11], [28] and an *optional* subset. The covering subset contains at least one replica of every data item, and servers in this subset (called *covering servers*) are never deactivated to ensure data availability. Servers in the optional subset (called *optional servers*) can then be deactivated without affecting data availability. Servers within each subset are arranged into a distinct ring with its own 127-bit keyspace. The system is configured with two parameters, one for the number of replicas (M) in the covering subset, and the second for replicas (O) in the optional subset. The M and O replicas of a data item are stored in the respective rings using the default Cassandra data placement algorithm.

Figure 2 shows this two-ring structure. Assuming a 27-server cluster, 9 covering servers and 18 optional servers, replication factors of 1 and 2, respectively, would distribute data evenly among the servers. This would also allow up to 2/3 of the servers to be deactivated without loss of data availability.

Server states. Each server in a GreenCassandra system can be in one of four states: *Active*–the server is active and handling client requests, *Off*–the server has been transitioned to a low-power state (e.g., ACPI's S3 state) by the GreenCassandra energy management policy, *Prepare*–the server is being brought up from the Off state and can serve write but not read requests, and *Failed*–the server has failed (independent of energy management). By definition, covering servers are always either Active or Failed.

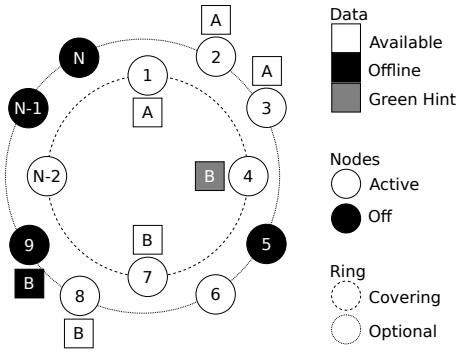


Figure 2. GreenCassandra's double-ring and data placement.

Green hints and server state transition. When a write affects data stored on an optional server in the Off state, we save a hint–called a *green hint* to differentiate from hints for handling server failures–on the local disk of a random non-owning covering server. If n owning optional servers are currently Off, n green hints will be sent to n distinct, randomly chosen, non-owning covering servers. *Green hints preserve data durability when optional servers are deactivated for energy management.*

When an Off server is reactivated, it first enters the Prepare state and queries all of the covering servers for green hints containing writes to its stored data. As noted above, this server will start servicing writes immediately (green hints overwritten by more recent writes are discarded). The server enters the Active state when it has processed all relevant green hints.

Consistency model. Let M be the number of replicas in the covering subset, T the number of replicas in the set of Prepare servers, A the number of replicas in the set of Active servers, and N the total number of replicas. (Note that $N \geq A \geq M$.) We modify the handling of reads as follows to preserve Cassandra's consistency model:

One: query any one of the Active owning servers.
Quorum: query $\lceil (A+1)/2 \rceil$ owning Active servers. This query set must include at least $\lceil (M+1)/2 \rceil$ owning servers in the covering subset.
All: query all A owning Active servers.

Correspondingly, we modify the handling of writes as follows. Each write is sent to N servers, where M of them are the owning covering servers, $A - M$ are Active owning optional servers, T are Prepare owning servers, and $N-A-T$ are random non-owning covering servers chosen to hold green hints for the Off owning servers. Then:

Any: wait for any forwarded write, green hint write, or the local write of a hint to complete.
One: wait for at least one forwarded write to an owning covering server to complete.
Quorum: wait for $\lceil (A+1)/2 \rceil$ forwarded writes to complete. This set of servers must include at least $\lceil (M+1)/2 \rceil$ owning covering servers.
All: wait for all N forwarded writes to complete.

The requirement for $\lceil (M+1)/2 \rceil$ owning covering servers targets consistency for Quorum and All (see Limitations below). Failures are handled in the same manner as Cassandra.

Limitations. The current design has two limitations: (1) it cannot guarantee strong consistency, and (2) it cannot tolerate more than $M - \lceil (M+1)/2 \rceil$ failures in the covering subset for write-Quorum, and, in some cases even for write-One. The first limitation arises because the transitioning of a server from Prepare to Active is not atomic with respect to the cluster; that is, for a short period of time, different nodes in the system can disagree on A, the number of Active nodes. This can lead to inconsistencies even when protocols meant for achieving strong consistency in Cassandra (e.g., Read-Quorum/Write-Quorum) are used in GreenCassandra. Note that this is also true for Cassandra if servers are added/removed while the system is running. This is why the original Yahoo Cassandra uses ZooKeeper to maintain global cluster information [25]. We can also adopt this approach to resolve this limitation. Currently, we limit the window of vulnerability by synchronizing server clocks (e.g., using NTP). Then, to become Active, a server will mark itself as Active at the current time plus that required for s rounds of gossiping has passed. s is the number of rounds needed to ensure that the change in the server's state has been propagated to all servers in the cluster with very high probability and is $\mathcal{O}(\log(clustersize))$ [29]. This is sufficient for our purposes, since our main evaluations are based on read-One/write-One (see Section V).

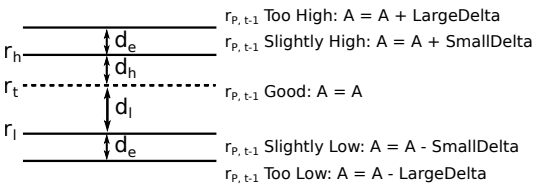
The second limitation can arise for write-One when M=1 since writes cannot complete when the owning covering server has failed. This limitation can be addressed by promoting appropriate servers from the optional subset when covering servers fail. This approach has been used in [13].

B. Power Management

Given the above power malleable architecture, we now describe the energy- and green-energy-aware policies that we have designed for power management in GreenCassandra. The policies seek to minimize brown electricity cost while respecting a performance SLA (P, D, S), where P percent of client requests received within the accounting time period S must be serviced with response time less than or equal to D. In the following text, we often refer to the P^{th} percentile response time as P^{th}-RT (e.g., 99^{th}-RT when P=99%), D as the SLA response time, and S as the SLA time period. All policies divide time into discrete epochs (e.g., 15 minutes), and determine the number of active servers for each epoch.

Reactive. We begin with an energy-aware (but green-energy-unaware) policy that is similar to PEGASUS [9]. This policy assumes that response time is being measured. It then adjusts the number of Active servers (A) to minimize energy consumption while conservatively trying to meet the SLA within each epoch. The policy also leaves some leeway for handling sudden spikes in the workload.

Specifically, let $r_{P,t-1}$ be the P^{th}-RT in epoch $t-1$ and r_t the target P^{th}-RT for this epoch. Reactive adjusts A as follows:



where $r_h \geq r_t \geq r_l$ and LargeDelta $\geq$ SmallDelta.

The intuition behind this policy is simple: set r_t conservatively below the SLA response time (not shown above), then make small adjustments if current performance is deviating from the target by a small but significant amount ($r_h + d_e > r_{P,t-1} > r_h$ or $r_l > r_{P,t-1} > r_l - d_e$), and larger adjustments if performance deviates by larger amounts ($r_{P,t-1} > r_h + d_e$ or $r_{P,t-1} < r_l - d_e$). Within each epoch, Reactive may also track an "instantaneous" P^{th}-RT (e.g., every 20 seconds). If this P^{th}-RT exceeds a threshold (r_e) for two checks, then Reactive increases A by a large amount ($EmDelta$) and restarts the epoch. This "emergency escape hatch" allows Reactive to react quickly to sudden load spikes.

Credit. In addition to being energy-aware, this policy uses excess green energy to build up performance slack, and then uses this slack later to reduce brown energy consumption. Specifically, Credit uses Reactive's adaptation approach with two differences: (1) it predicts green energy production one epoch into the future, and, if it expects excess energy, it will activate as many servers as can be powered by the expected green energy, and (2) it tracks the cumulative performance thus far within the SLA accounting time period, and adjusts the target P^{th}-RT for the next epoch if performance slack has been accumulated. For example, after a period with high green energy production, GreenCassandra may have serviced 99% of the load thus far within 60ms compared to an SLA of (99%, 75ms, 1day). At the start of the next epoch, if it expects that brown energy will be used, Credit then may set the target 99^{th}-RT higher than 75ms (e.g., 100ms).

Credit uses accumulated slack slowly to avoid epochs with spikes in response time. Continuing the example above, regardless of how much performance slack has been accumulated, Credit will not target a 99^{th}-RT any higher than a threshold value for any epoch. Credit uses a simple power model, assuming that each active server consumes a constant amount of power, to compute the number of active servers that can be powered by the expected solar energy production.

Credit-M. This policy accumulates and uses performance slack for reducing brown energy consumption in the same manner as Credit. However, it assumes a performance model $N(p, r, l)$ that specifies the number of active servers needed to ensure that p percent of requests can be completed within r time when the offered load is l. It then predicts the workload and solar energy production in the next epoch, and uses the performance model and the target P^{th}-RT to compute the base number of active servers. (We discuss the construction of all models required by Credit-M and Opt, the next policy, and techniques for workload and solar energy prediction in Section IV-C)

Opt. This policy assumes knowledge (possibly through prediction) of the load and green energy production in a scheduling horizon T that evenly divides the SLA time period S. It then solves an optimization problem to compute the number of Active and Prepare servers for each time epoch in T. Table I describes the parameters of our optimization framework.

Minimizing energy cost. The optimization problem is to minimize the brown electricity cost:

$$Cost = \sum_{t \in T} BEnergy(t) \cdot BEnergyPrice(t) \qquad (1)$$

Table I. PARAMETERS OF THE OPTIMIZATION FRAMEWORK.

Inputs	**Description**
N	The number of GreenCassandra servers
T	Scheduling horizon
$BEnergyPrice(t)$	Price for brown energy in epoch t
$GEnergy(t)$	Green energy produced in epoch t
$L(t)$	Load in epoch t
$MPRT$	The max P^{th}-RT allowed for any epoch
Outputs	**Description**
$Act(t)$	Number of Active servers in epoch t
$Prep(t)$	Number of Prepare servers in epoch t
$BEnergy(t)$	Brown energy needed in epoch t

Assuming a power model $Power(a, x, l)$ that gives the power required to service a load l with a Active nodes and x Prepare nodes, the energy used in each time epoch t is:

$$Energy(t) = Power(Act(t), Prep(t), L(t)) \cdot |t| \qquad (2)$$

and the amount of brown energy needed during an epoch t is:

$$BEnergy(t) = Energy(t) - GEnergy(t) \qquad (3)$$

Modeling node transitions. When transitioning a server from Off to Active, the server must first be brought up-to-date while in the Prepare state. In general, the amount of time that a server needs to be in the Prepare state depends on the load offered to the system during the time it has been Off. However, in all of our evaluation, updating a server never takes longer than one epoch used in Opt (15 minutes). Further, we can limit the update time by capping the amount of time that any server can be Off. Thus, for simplicity, we model this transition by requiring a node to be in the Prepare state one epoch before it becomes Active.

We have also developed an extended optimization problem that includes variable time for servers in the Prepare state, and a heuristic-based approach for solving the problem. We do not present this extension here because of space constraints.

Constraints. We define a performance constraint to ensure that the performance SLA (P, D, S) is met. Let $CDF(t, r)$ be the fraction of client requests serviced within time r in epoch t. Then, the performance SLA will be met if:

$$\frac{\sum_{t \in T} CDF(t, D) \cdot L(t)}{\sum_{t \in T} L(t)} \geq \frac{P}{100\%} \qquad (4)$$

Assuming a performance model $F(a, x, r, l)$ that gives the fraction of client requests serviced within time r, where a is the number of Active servers, x is the number of Prepare servers, and l is the load, Equation 4 becomes:

$$\frac{\sum_{t \in T} F(Act(t), Prep(t), D, L(t)) \cdot L(t)}{\sum_{t \in T} L(t)} \geq \frac{P}{100\%} \qquad (5)$$

Finally, we add a constraint for capping the worse allowable P^{th}-RT in any epoch ($\forall_t, CDF(t, MPRT) \geq P$) to avoid unacceptable short-term performance degration.

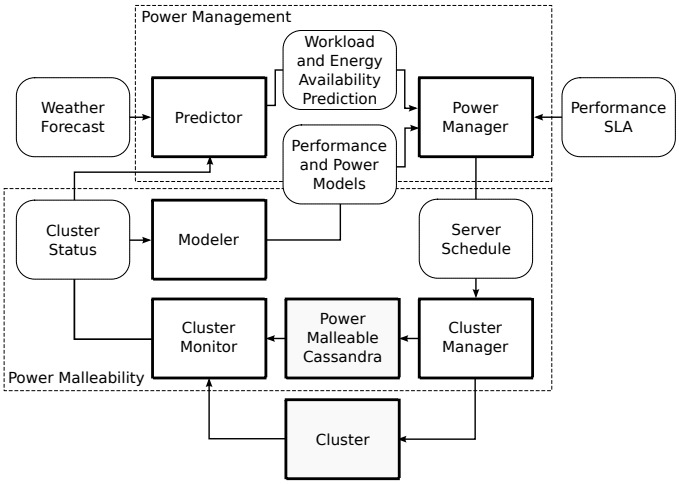


Figure 3. GreenCassandra's architecture.

Solving the optimization problems. The above optimization problem can be expressed as a Mixed Integer Linear Programming (MILP) problem. This problem can be efficiently solved using solvers such as Gurobi [30] for reasonably sized clusters; e.g., 20s on average for our 27-server cluster. In fact, since the optimization problem can be solved during the epoch before the solution is adopted, its running time even for larger systems has little impact as long as it is less than the epoch time. Opt tracks system performance and predictions of workload and green energy production every epoch, and rerun the optimization if it detects significant deviations.

C. Prototype

We have implemented a prototype of GreenCassandra that extends Cassandra 1.1.6 for Linux with roughly 4,000 lines of Python wrapper code, and adds or modifies around 3,400 lines in Cassandra itself. Figure 3 shows the major components in the implementation. Note that this general architecture is used to implement all policies described in Section IV-B, although not all policies use all components.

The **Cluster Monitor** tracks system performance and energy consumption. We use the existing monitoring infrastructure of our experimental platform (see Section V).

The **Modeler** uses the monitored data to produce/adapt the performance and power models required by Credit-M and Opt. While both models depend on the number of servers in the Active and Prepare state, our implementation stages the catching-up of Prepare servers to remove the latter dependency. Then, to build the performance model, we measure the response time CDF for multiple different combinations of number of Active servers and load intensity using a representative trace of the workload. We use the measured values to construct table lookup models with interpolation between measure points. Figure 4 shows a slice of the model for Credit-M. Note that while we would expect the construction and adaptation of the performance models over time to be automated in a production system, we have not done so at this point; i.e., we have manually built the models for our evaluation.

We use a simple power model, with each server drawing a constant (but different) amount of power in each state. We have also built models with load-dependent power demand for

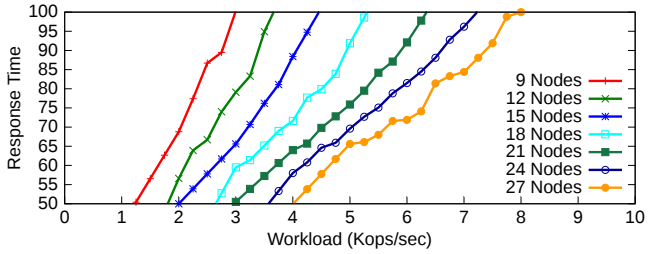


Figure 4. Model slice for the 99^{th}-RT.

servers in the Prepare and Active states, but found that they did not provide tangible benefits because the dynamic power demand of the servers in our experimental platform is small.

The **Predictor** implements the needed load and solar energy prediction. We implement near-future load prediction (for Credit-M) as follows. During each epoch, we measure the offered load across several subintervals. Then, at the end of the epoch, we calculate a function using linear regression. We extend this regression line into the future, sanity check it, and then use it to predict the load for the next epoch. How far we extrapolate into the future (one vs. several epochs) is workload dependent (see Section V)

While previous work has shown that some interactive workloads can be predicted days into the future with high accuracy (e.g., [20]), we have not implemented the long-term load prediction needed to make Opt a practical on-line algorithm. In our evaluations below, we instead run Opt with perfect knowledge of future load and green energy production to bound the best achievable energy consumption and cost.

We use the method from [16] to predict solar energy production. This method combines the model from [31] with the approach for improving accuracy from [15]. Briefly, the model relates solar energy generation to cloud cover as $E_p(t) = B(t)(1 - CloudCover(t))$, 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 $CloudCover(t)$ is the forecasted percentage cloud cover. We use forecasts from Intellicast.com, which predicts hourly cloud cover 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, leading to inaccurate predictions. Thus, the prediction includes a technique for using recent energy production to predict future production when predictions based on weather forecasts are observed to be inaccurate [15]. Previous work has shown that the overall prediction approach can predict solar energy production with sufficient accuracy 48 hours into the future [16].

The **Power Manager** implements our energy management policies (although it runs only one at a time), while the **Cluster Manager** transitions servers into/out-of ACPI S3 to deactivate/activate them. We use ACPI S3 instead of turning servers completely off for fast transitioning between states.

The **Power Malleable Cassandra** is our modified Cassandra. In addition to techniques already described, we found it necessary to carefully control compaction to keep response time stable. We thus configured our modified Cassandra to

use "leveled compaction," which performs frequent small compactions instead of larger ones. We also carefully control the number of servers that are simultaneously compacting and the maximum allowed compaction rate on each server.

V. Evaluation Methodology

Experimental platform. We run experiments on a 27-server cluster in Parasol, a micro-datacenter partially powered by solar energy [4]. Each machine has two 1.8GHz Atom processor cores, 4GB of RAM, a 500GB hard disk, and a 64GB solid-state disk. Each server consumes 22-30 Watts and 3 Watts when active and in ACPI S3, respectively. Transitioning into/out-of ACPI S3 takes a total (round-trip time) of 7secs. The servers are inter-connected by a Gigabit Ethernet switch. The client workload is emulated using a separate server.

Parasol includes an extensive monitoring infrastructure that can measure the power draw of each server as well as the power drawn from the grid and the solar system. Thus, all reported energy consumption are measured quantities. The single exception is the power draw from the solar system, since we simulate the solar energy production in our experiments for proper scaling and repeatability.

Workloads. We emulate three workloads using three traces: Ask.com, MS Hotmail, and MS Messenger. Ask.com is a 1-month trace of the number of requests per second arriving to a slice of the Ask.com search engine during April 2008. The Hotmail and Messenger traces contain normalized aggregated disk I/O request rates for one week for the respective service [13]. We use each trace to generate a workload with time-varying load intensity in requests per second.

We use a customized version of the Yahoo Cloud Services Benchmark (YCSB) [32] to generate client requests. We extended YCSB to support (1) a Poisson request arrival process, and (2) load balancing under dynamic cluster size scaling by integrating the Hector Cassandra client [33] into YCSB, while modifying Hector to handle regular changes in cluster membership. For each workload, we match the average request rate to the traces for each 30 second interval. For request generation, we use the default YCSB Zipf distribution with a 95:5 read/write ratio. Requests use Read-One/Write-One, implying a weak consistency model; we choose this setting because there is strong evidence showing that Cassandra is used in this mode in the industry [34].

To scale the workloads to our cluster, we use our performance model (see below) to compute the maximum request rate that can be serviced with the full-size cluster, while meeting the performance SLA. The peak load of each workload is then set to 75% of this maximum rate, and the rest of the workload is normalized accordingly. Most of our experiments are run with a performance SLA of (99%, 75ms, 1day). The 75ms is chosen to fit within the overall response time targets of a couple of hundred ms for many typical Web/cloud services.

Figure 5 plots the resulting full-size cluster utilization for the workloads over a 1-week period. Our evaluations use a representative weekday (Monday) within this period. Observe that while all three workloads show clear diurnal patterns (which simplifies workload prediction), each pattern has different attributes that differentiate them from each other.

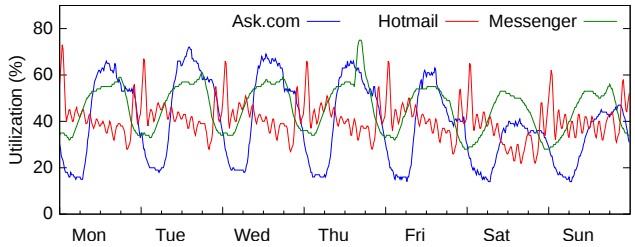


Figure 5. System utilization for each workload.

Messenger has a large peak early each day, but is relatively flat for the remainder of the day. Ask.com and Hotmail are similar, but Hotmail has lower peaks and higher valleys, and the peaks are later in the afternoon.

The short-term load prediction approach described in Section IV-C is accurate for all workloads although Hotmail requires the furthest extrapolation (several epochs) into the future because of the large load spike early in the day.

Parameterizing policies. We study two parameterization of Reactive, a conservative one called Reactive (r_l:56.25ms, r_t, r_h:63.75ms, r_e:75ms, d_e:5ms, $SmallDelta$:1, $LargeDelta$:3, $EmDelta$:5 for Ask.com) and a more aggressive one called Reactive-A (r_l:67.5ms, r_t, r_h:75.0ms, d_e:10ms, $SmallDelta$:1, $LargeDelta$:3). The former targets constantly preserving the SLA response time, while the latter may allow performance to be worse than the SLA response time in some short time periods, but preserves the SLA within its overall accounting period. Reactive-A also does not use the emergency escape hatch. All parameters were derived manually. Credit uses the same base parameters as Reactive-A.

We constrain Credit, Credit-M, and Opt to never target a 99^{th}-RT worse than 100ms.

Finally, we run Reactive, Reactive-A, and Credit using 5-minute epochs, and Credit-M and Opt using 15-minute epochs. Credit-M and Opt can tolerate longer epochs because they are predictive as opposed to reactive.

Solar energy production. Parasol's solar array can produce up to 3.2kW of AC power (after derating). For repeatability, we use traces of solar energy production from several days. We also scale down the solar energy production because we are using less than a quarter of the servers that can be housed in the datacenter (Parasol is not yet fully provisioned with servers). The peak production of the scaled system is just enough to power all servers running at maximum power draw. We use scaled down traces from three days, one with high (10/20/2013), one with medium (9/29/2013), and one with low (10/11/2013) solar energy production. We call these days High, Medium, and Low, respectively.

Brown electricity pricing. Datacenters often contract with their power companies to pay variable brown energy prices. The most common arrangement is for the datacenter to pay more for brown energy consumed during an on-peak period (e.g., daytime) than during an off-peak period (e.g., at night). Correspondingly, we use on-peak/off-peak pricing with PSEG New Jersey prices: \$0.13/kWh for on-peak (9am-11pm) and \$0.08/kWh for off-peak (11pm-9am).

Accelerating and validating the experiments. Each of our

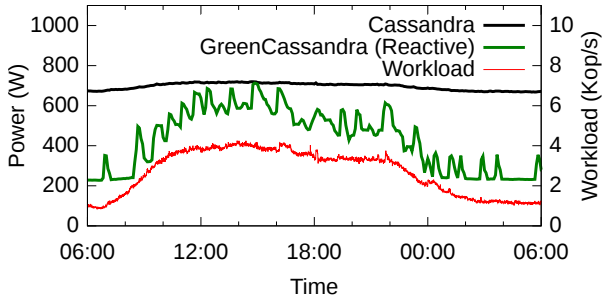


Figure 6. The Ask.com workload (right y-axis) during day 1 of the week (Monday), and the power demand for Cassandra and Reactive GreenCassandra (left y-axis).

experiments covers a period of 24 hours to explore a complete cycle of solar energy generation. To reduce the experimental time, we speed up each experiment 4 times (i.e., we compressed the workload patterns and solar energy generation by a factor of 4). Thus, each 15 minutes of our experimental run corresponds to one hour of real time.

VI. Evaluation Results

This section presents our experimental results for GreenCassandra. We begin by briefly comparing Reactive against unmodified Cassandra to show the impact of being energy-aware. We then compare Reactive, Reactive-A, Credit, Credit-M, and Opt to study the benefits of more aggressive parameterizations, being green-aware, having more information about the system (i.e., performance model), and having oracular knowledge of the future. Finally, we consider the impact of different amounts of solar energy and different workloads on the policies.

Impact of being energy-aware. Figure 6 plots the Ask.com workload for 24 hours, together with the power demand for Reactive and Cassandra when running this workload. The SLA is (99%, 75ms, 1day). Observe that Cassandra's power demand is nearly constant since it never deactivates any server, and the dynamic power demand of our cluster is small compared to the static base demand. In contrast, Reactive adjusts the number of active servers to reduce power demand while preserving the performance SLA. By the end of the day, Reactive reduces the energy consumption by 48% by trading some performance, achieving a 99^{th}-RT of 58ms compared to 39ms for Cassandra. Being conservative, Reactive never allows the short-term 99^{th}-RT, defined as the 99^{th}-RT of a 20-minute sliding window measured every 5 minutes, to rise above the SLA response time (Figure 7(b)). (In fact, its reactive nature and conservative emergency response causes the power demand to be rather "spiky.") We conclude that energy-aware GreenCassandra can significant reduce energy consumption compared to Cassandra, and thus, in the remainder of the section, we will use Reactive as the comparative baseline to assess the benefits of other policies in GreenCassandra.

Impact of more aggressive power reduction and being green-energy-aware. Next, we compare Reactive, Reactive-A, and Credit to assess the benefits of (1) allowing the short-term response time to sometime exceed the SLA response time (while still observing the SLA within the specified time period) for reduced energy consumption, and (2) being green-energy-aware. Figure 7(a) plots the solar power production and the

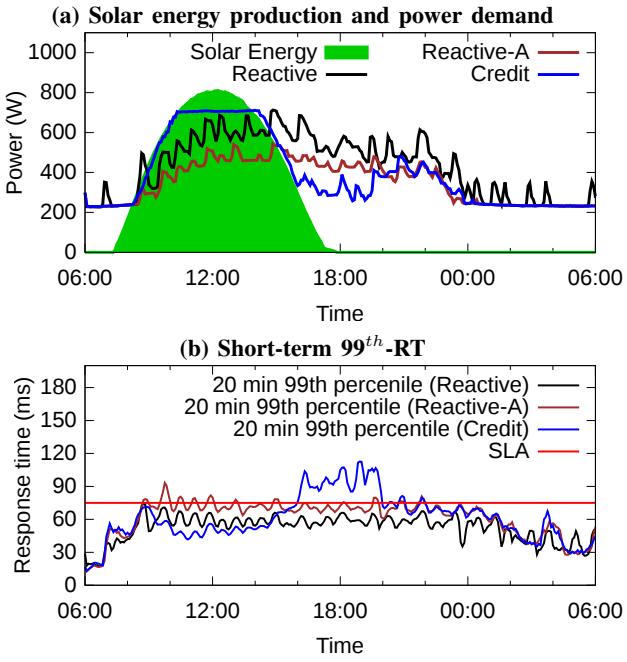


Figure 7. Comparing Baseline, Reactive, and Credit for the High day.

power demand of Reactive, Reactive-A, and Credit for the High day. Figure 7(b) plots the short-term 99^{th}-RT for the three policies.

First, observe that Reactive-A does allow the short-term 99^{th}-RT to sometimes exceed the SLA response time between 8am-4pm and also briefly close to 10pm. However, the short-term 99^{th}-RT never exceeds 100ms, and, by the end of the day, the cumulative 99^{th}-RT response time is 69ms, which meets the SLA. This allows Reactive-A to reduce both brown energy consumption and cost by 19% and 20%, respectively. Interestingly, Reactive-A also uses less green energy than Reactive because it is oblivious to green energy availability.

Second, observe that Credit aggressively turns on more servers than is needed when green energy is available to build up performance slack ($\sim$9am-3pm). This slack is then used from $\sim$4pm-7pm to avoid or reduce brown energy consumption. Similar to Reactive-A, Credit allows the short-term 99^{th}-RT to rise above the SLA response time; however, because Credit caps the amount of slack that can be used in any epoch, the short-term 99^{th}-RT never degrades further than 120ms, with most remaining below 90ms. By the end of the day, Credit achieves a 99^{th}-RT of 70ms.

Compared to Reactive, Credit increases green energy consumption by 15%, and reduces brown energy consumption and cost by 26% and 28%, respectively. Compared to Reactive-A, Credit increases green energy consumption by 37%, and reduces brown energy consumption and cost by 8% and 10%, respectively. These results demonstrate that Credit can beneficially shift energy demand to better match green energy production, while still meeting the SLA.

Impact of oracular knowledge of future load and green energy production. Finally, we compare Opt with Credit to assess the benefits of using accurate future load and green-energy production information. Figure 8 plots the power demand and short-term 99^{th}-RT for Credit and Opt. Observe that

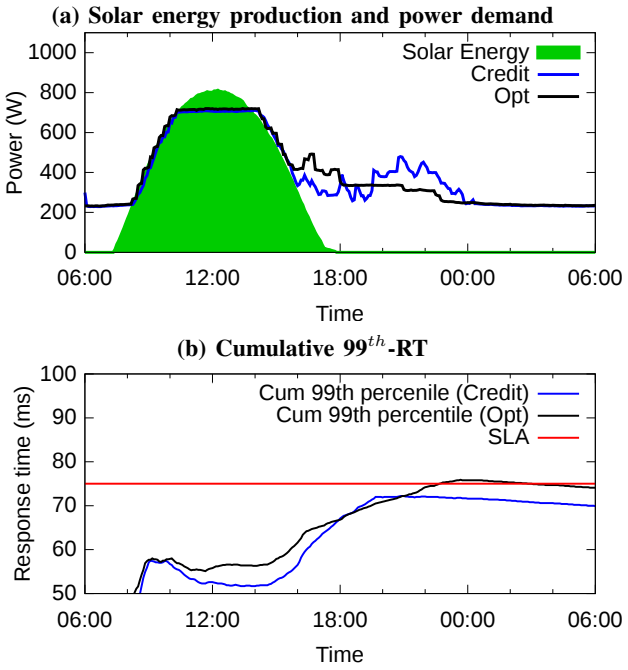


Figure 8. Comparing Credit and Opt for the High day.

the two policies perform very similarly. Overall, Opt reduces brown energy consumption by 1% and increases green energy consumption by 1%. Opt slightly outperforms Credit because it is able to more aggressively reduce brown energy consumption between ∼10pm-2am, where it allows the cumulative 99^{th}-RT to rise above the SLA response time, knowing that it can recover this performance loss from ∼3am-6am, when there will be more Active servers than needed because GreenCassandra cannot deactivate servers in the covering subset. Credit is not able to do this because it never allows the cumulative 99^{th}-RT to exceed the SLA response time.

Interestingly, Opt's power demand is much smoother over time than Credit. This is because Opt knows the future load and green energy production, and accounts for the energy consumed by servers in the Prepare state, all of which lead to fewer adjustments. In contrast, some of Credit's actions may need to be reversed in the near future because of changing conditions, leading to some unnecessary adjustments.

Impact of using short-term load prediction and performance model. While we do not show the results here, Credit and Credit-M perform very similarly. Power demand is somewhat smoother under Credit-M, but the difference is smaller than that between Opt and Credit. Thus, the advantage of Credit-M compared to Credit is that the adjustment parameters do not have to be determined experimentally. The disadvantage, of course, is that the performance model has to be constructed; however, this process can be automated. Both Credit's parameters and Credit-M's model may have to be adjusted over time.

Sensitivity to green energy availability. In the previous experiments, we have considered a sunny day with a high amount of solar energy production. We now consider the impact of days with lower energy production. Figure 9 plots the green energy production, power demand, and short-term 99^{th}-RT for Credit and Opt on the Medium solar day (Reactive is the same since

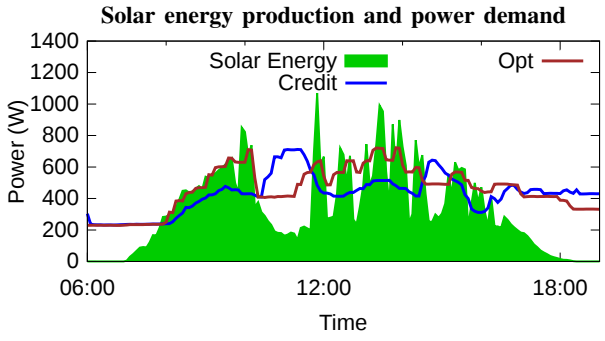


Figure 9. Comparing Credit and Opt for the Medium day.

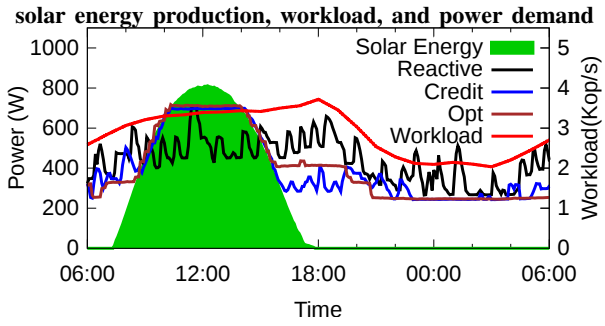


Figure 10. Comparing Reactive, Credit, and Opt for the messenger workload on high solar energy day.

it does not consider green energy production). In this case, because the periods of solar energy production mostly correlate with periods of high load, Reactive can already consume most of the solar energy produced. Credit actually consumes slightly less green energy (7%), while Opt consumes more green energy (9%) than Reactive. Opt's advantage over Credit arises from the fact that it is more difficult to predict solar energy production accurately for days with fluctuating cloud cover such as Medium. Opt's perfect knowledge of green energy production allows it to more efficiently use the green energy to gain performance slack between ∼12noon-3pm, which it then uses between ∼6-10pm to reduce brown energy consumption. Overall, Credit and Opt reduce brown energy consumption by 18% and 23%, respectively, compared to Reactive. They reduce cost by 19% and 24% compared to Reactive.

On the Low solar day, Credit and Opt are essentially the same, and consume less brown energy than Reactive (19% and 18%, respectively) because they bound the number of active servers more tightly while still meeting the SLA.

Sensitivity to different workload characteristics. Finally, we consider the impact of different workload characteristics. Figure 10 plots the solar energy production, the Messenger workload, and the power demand of the three policies for the High day. In this case, the load during periods of high solar energy production is lower compared to the Ask.com workload, allowing Credit and Opt to accumulate more performance slack compared to Reactive. This leads to larger reductions in brown energy consumption, 28% and 29%, and cost, 29% and 29%, for Credit and Opt, respectively. Results are similar for the Hotmail workload, with Credit and Opt reducing brown energy consumption by 24% and 26% and cost by 25% and 27%, respectively.

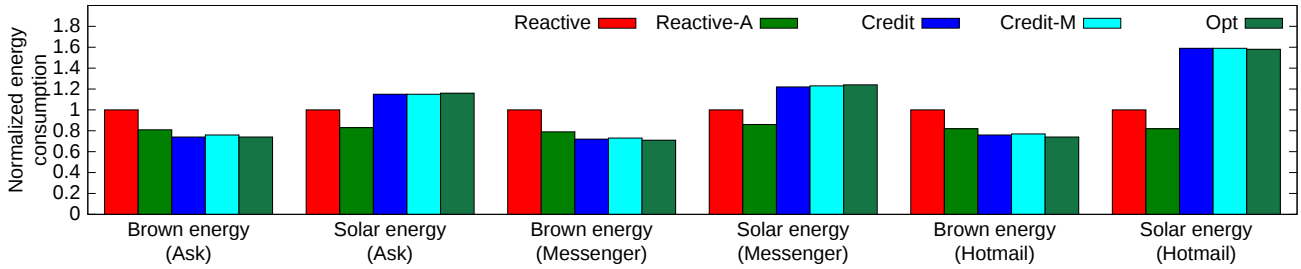


Figure 11. Normalized brown and green energy consumption for the three workloads running on the High day.

VII. Conclusions

In this paper, we proposed GreenCassandra, a green-energy-aware distributed structured storage system that leverages self-generation of solar energy to reduce brown energy consumption and cost, while meeting a performance SLA. We proposed, implemented, and evaluated several power management policies. Figure 11 summarizes our findings, showing that green-energy-aware policies can significantly reduce brown energy consumption and cost compared to a power-aware but green-energy-unaware baseline policy; e.g., Credit reduces brown energy consumption on a day with high solar energy production by 26%, 28%, and 24% for the Ask.com, Messenger, and Hotmail workloads, respectively, compared to Reactive. Further, the savings for the heuristic-based policies (Credit and Credit-M) are very close to those of a policy that has perfect future information for green energy production and load (Opt). We thus conclude that we can shift power demand for an interactive workload to when green energy is available to reduce brown energy consumption. We believe that our work can be generalized to other cluster-based interactive services since Cassandra is a complex system that services both read and write requests.

References

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

[2] A. Venkatraman, "Global census shows datacentre power demand grew 63% in 2012," Computer Weekly, October 08, 2012.

[3] International Electrotechnical Commission, "Efficient Electrical Transmission and Distribution," Tech. Rep., 2007.

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

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

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

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

[8] GreenQloud, 2013, http://greenqloud.com".

[9] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards Energy Proportionality for Large-Scale Latency-Critical Workloads," in *ISCA*, 2014.

[10] A. Krioukov *et al.*, "Design and Evaluation of an Energy Agile Computing Cluster," University of California at Berkeley, Tech. Rep. EECS-2012-13, 2012.

[11] E. Pinheiro, R. Bianchini, and C. Dubnicki, "Exploiting Redundancy to Conserve Energy in Storage Systems," in *Proceedings of SIGMETRICS/Performance*, 2006.

[12] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan, "Robust and Flexible Power-Proportional Storage," in *SoCC*, 2010.

[13] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: Practical power-proportionality for data center storage," in *Eurosys*, 2011.

[14] N. Sharma, D. Irwin, and P. Shenoy, "A Distributed File System for Intermittent Power," in *IGCC*, 2013.

[15] 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 *SC*, 2011.

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

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

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

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

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

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

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

[23] S., R. Sohan, A. Rice, A. Moore, and A. Hopper, "Free Lunch: Exploiting Renewable Energy for Computing," in *HotOS*, 2011.

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

[25] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," vol. 44, no. 2, 2010.

[26] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "BigTable: A Distributed Storage System for Structured Data," in *OSDI*, 2006.

[27] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," in *SOSP*, 2007.

[28] J. Leverich and C. Kozyrakis, "On the Energy (In)Efficiency of Hadoop Clusters," in *HotPower*, 2009.

[29] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient reconciliation and flow control for anti-entropy protocols," in *LADIS*, 2008.

[30] Gurobi Optimization, Inc., "Gurobi Optimizer 5.6," 2013, http://www.gurobi.com.

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

[32] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC*, 2010.

[33] "Hector: A high level Java client for Apache Cassandra," 2014, http://hector-client.github.io/hector/build/html/index.html.

[34] Christos Kalantzis, "A Netflix Experiment: Eventual Consistency != Hopeful Consistency," 2013, http://planetcassandra.org/blog/a-netflix-experiment-eventual-consistency-hopeful-consistency-by-christos-kalantzis/.