GreenCassandra: Using Renewable Energy in Distributed Structured Storage Systems*

William Katsak†, Íñigo Goiri‡, Ricardo Bianchini†‡, Thu D. Nguyen†

†Department of Computer Science, Rutgers University
{wkatsak, ricardob, tdnguyen}@cs.rutgers.edu

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

Published in Proceedings of the International Green and Sustainable Computing Conference (IGSC), Nov 2015

***Abstract*—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 variable sources such as solar and wind. 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, compared to a baseline energy-aware policy. 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 [1], [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. For example, Apple has built two 40MW solar arrays next to their datacenter in Maiden, NC [3]. Green House Data [4], AISO [5], and GreenQloud [6] 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-deferrable workload*.

We have designed three energy- and green-energy aware scheduling policies for GreenCassandra, targeting datacenters partially powered by solar energy.[2] 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 a real solar-powered datacenter. Our evaluation uses three workloads, each constructed to follow a trace of a real system (Ask.com, MS Hotmail, and MS Messenger).

*This work was partially supported by NSF grant CSR-1117368 and the Rutgers Green Computing Initiative.

[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 [7].

[2] Except for the prediction of solar energy production, our work is directly applicable to wind energy as well.

For repeatability and proper scaling, our evaluation uses scaled-down traces of solar energy production from the datacenter. 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 deferrable); 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.* [8] 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.* [9] 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 [10]–[12]. The data layout of GreenCassandra is inspired by [10]. Amur *et al.* [11] did not study energy management policies, while Pinheiro *et al.* [10] and Thereska *et al.* [12] did not consider green energy. Further, they did not target maintaining a response time SLA.

Previous works have proposed Blink [13] and BlinkFS [14] as systems that can adjust their power demand to match intermittent power production. BlinkFS, the storage system, 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 [7], [9], [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.

It may be possible to implement GreenCassandra in the GreenSwitch framework [7]. However, this previous work studied a MapReduce system, rather than an interactive system such as GreenCassandra.

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 ours 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 policies can also trade periods where short-term performance can degrade beyond the SLA for lower energy consumption/cost.

III. Background: 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).

Cassandra uses a tunable consistency model based on a client-specified number of replicas that must be read/written

for a request to successfully complete. Specifically, a read request can be serviced by *one*, a *quorum*, or *all* owning servers. A write request can complete once it has been completed at *any*, *one*, a *quorum*, or *all* owning servers. *Any* is different from *one* in that a completed write of a hint is sufficient, whereas *one* requires that a write must complete at one of the owning servers. Keep in mind that each write is forwarded 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. Writes are serialized using a user-provided timestamp. Each replica of a data item stores the timestamp of the last write.

IV. Structured Storage in Green Datacenters

We propose GreenCassandra, a distributed structured storage system for datacenters powered by PV solar panels and the grid.[3] 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., [10], [12], [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 [10], [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 1 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.

[3]The design we study does not include batteries or net metering for the reasons mentioned in Section I.

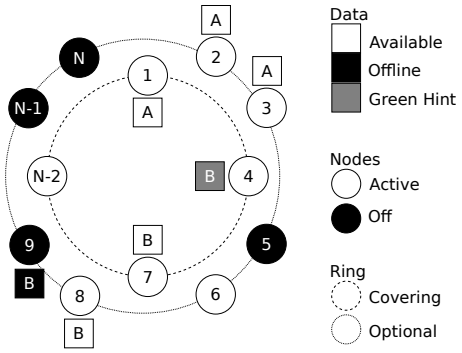


Figure 1. GreenCassandra's double-ring and data placement.

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 forwarded 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.

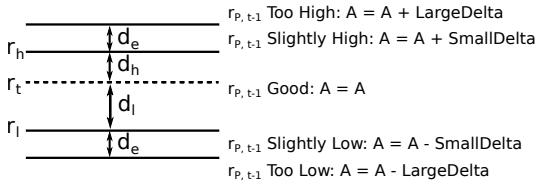
Limitations. The current design has two limitations: (1) it cannot guarantee strong consistency, and (2) it cannot tolerate more than $M - \lceil (M+1)/2 \rceil$ failures in the covering subset for write-Quorum, and, in some cases even for write-One. The first limitation arises because the transitioning of a server from Prepare to Active is not atomic with respect to the cluster, and can be addressed by using a coordination mechanism such as ZooKeeper [29]. The second limitation can arise for write-One when M=1 since writes cannot complete when the owning covering server has failed, and can be addressed by promoting appropriate servers from the optional subset when covering servers fail (e.g., [12]).

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 [8]. 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.

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

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:

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

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.

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:

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

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:

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

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.

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. Logically, this prototype comprises a number of components: **Cluster Monitor**, **Modeler**, **Predictor**, **Power Manager**, and **Power-Malleable Cassandra**.

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 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 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. 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 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 solar energy prediction. We use the method from [16] to predict solar energy production using weather forecasts. The evaluation in [16] shows that one-hour ahead predictions using this method produce inaccuracies of around 11%. GreenCassandra can easily tolerate this level of inaccuracies as shown in Section VI.

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, implementing the techniques described in Section IV-A.

V. Evaluation Methodology

Experimental platform. We run experiments on a 27-server cluster in a datacenter partially powered by solar energy. 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.

The datacenter 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. We do not consider GreenCassandra's impact on datacenter cooling.

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 [12]. 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) [31] to generate client requests. We extended YCSB to support (1) a Poisson request arrival process, (2) load balancing under dynamic cluster size scaling, and (3) adjustment of target request rate in real time. We also added an external workload controller that continually adjusts YCSB's target request rate according to a given workload trace.

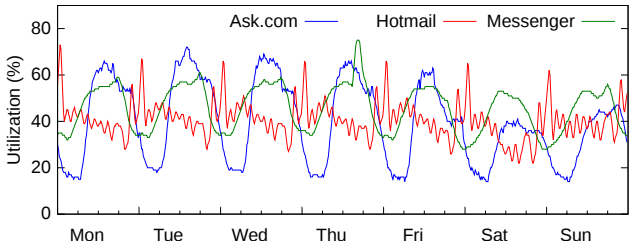


Figure 2. System utilization for each workload.

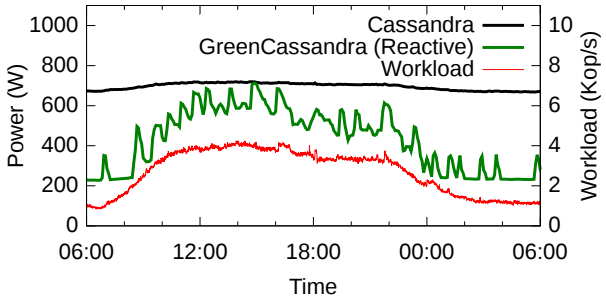


Figure 3. The Ask.com workload (right y-axis) during day 1 of the week (Monday), and the power demand for Cassandra and Reactive GreenCassandra (left y-axis).

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 [32].

To scale the workloads to our cluster, we use our performance model to compute the maximum request rate that can be serviced with the full-size cluster, while meeting the performance SLA. The peak load in each trace is set to 75% of this maximum rate, and the rest of the trace is normalized accordingly. The resulting trace is then used as input to the workload controller mentioned above. Most of our experiments are run with a performance SLA of (99%, 75ms, 1day)[4]. 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 2 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. 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.

Parameterizing policies. We study two parameterizations 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: 75ms, 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.[5] Credit uses the same base parameters as Reactive-A. We constrain Credit 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 Opt using 15-minute epochs. Opt can tolerate longer epochs because it is predictive as opposed to reactive.

Solar energy production. The solar array of the experimental datacenter 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 (the datacenter 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 prices from the utility in our state: \$0.13/kWh for on-peak (9am-11pm) and \$0.08/kWh for off-peak (11pm-9am).

Accelerating the experiments. Each of our 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, and Opt to study the benefits of more aggressive parameterizations, being green-aware, 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 3 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

[4] An SLA period of several hours should be sufficient for GreenCassandra to realize some of its savings using solar energy (see Section VI), with decreasing savings as the SLA period becomes smaller.

[5] In general, it may not always be easy to derive these parameters manually. This challenge increases the attractiveness of Opt when it is possible to predict workload accurately.

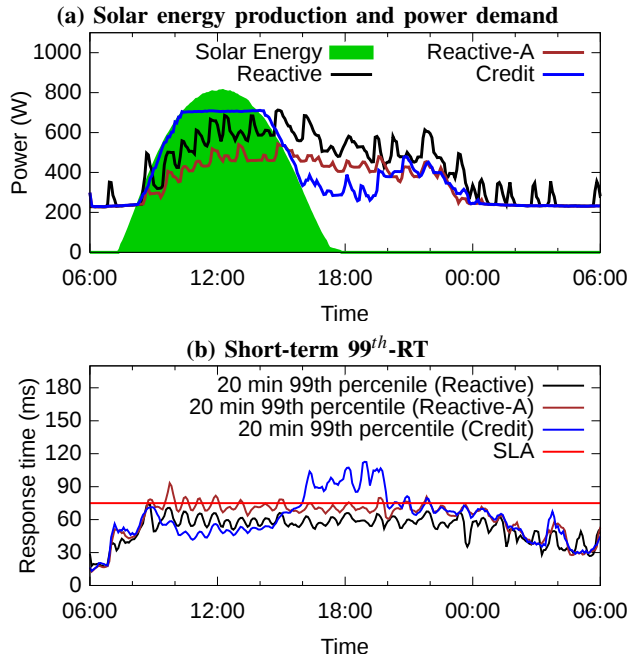


Figure 4. Comparing Baseline, Reactive, and Credit for the High solar day.

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 4(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 sometimes 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 4(a) plots the solar power production and the power demand of Reactive, Reactive-A, and Credit for the High day. Transferring green hints to nodes in the Prepare state can increase write traffic in the cluster by 28–51% (but less than 1% increase in overall data transfer). Server power demand for all state transitioning activities is included. The brown power demand is given by the difference between the power demand and solar power production. Figure 4(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 (∼9am-3pm). This slack is then used from ∼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. In fact, the short-term 99^{th}-RT sometime degrades slightly beyond 100ms, the maximum allowed target for Credit, because of Credit's aggressive parameterization. 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. (We do not show the results because of space constraints.) 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 at night. It is able to do so because its perfect knowledge of the future allows it to predict that it can recover the lost performance later. Credit is not able to do this because it cannot know the future, and therefore never allows the cumulative 99^{th}-RT to exceed the SLA response time. Further, 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.

Sensitivity to green energy availability. On the Medium solar day, 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. Credit's difficulty arises from mis-predictions of solar energy production because the Medium day has fluctuating cloud cover. 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. In the case of the Messenger workload, 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.

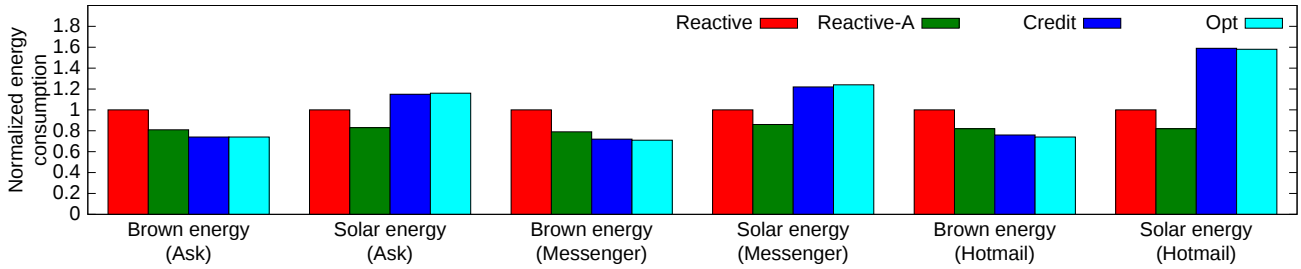


Figure 5. Normalized brown and green energy consumption for the three workloads running on the High day.

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.

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

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

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

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

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

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

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

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

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

[12] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: Practical Power-proportionality for Data Center Storage," in *Eurosys*, 2011.

[13] N. Sharma, S. Barker, D. Irwin, and P. Shenoy, "Blink: Managing Server Clusters on Intermittent Power," in *ASPLOS*, 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, J. Venkatesh, L. Zhang, and T. Rosing, "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, M. Lin, A. Wierman, S. Low, and L. Andrew, "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," *SIGOPS Operating Systems Reviews*, 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] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *USENIX ATC*, 2010.

[30] Gurobi Optimization, Inc., "Gurobi Optimizer 5.6," 2014, http://www.gurobi.com.

[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *SoCC*, 2010.

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