CADRE: Carbon-Aware Data Replication for Geo-Diverse Services

Zichen Xu
The Ohio State University
xu.925@osu.edu

Nan Deng
The Ohio State University
dengn@cse.ohio-state.edu

Christopher Stewart
The Ohio State University
cstewart@cse.ohio-state.edu

Xiaorui Wang
The Ohio State University
wang.3596@osu.edu

***Abstract*—Internet services replicate data to geo-diverse sites around the world, often via consistent hashing. Collectively, these sites span multiple power authorities that independently control carbon emissions at each site. Serving data from a carbon-heavy site increases the service's carbon footprint, but it is hard to place data at sites that will have low emission rates without replicating to too many sites. We present CADRE, a carbon-aware data replication approach. CADRE forecasts emission rates at each site and replicates data to sites that combine together to yield low carbon footprints. It makes replication decisions online, i.e., when data is created, and thus avoids emissions caused by moving data frequently in response to changing emission rates. CADRE uses the multiple-choice secretary algorithm to replicate objects with large footprints to low emission sites. It models carbon footprints for each object using the *footprint-replication curve*, a graph that maps replication factors to expected carbon footprints. CADRE also achieves availability goals, respects storage capacity limits and balances data across sites. Compared to consistent hashing, our approach reduces carbon footprints by 70%. It also supports and enhances the state-of-the-art green load balancing, reducing the carbon footprint by an additional 21%.**

I. Introduction

Over the next 5 years, the number of Internet users will grow by 60% and Internet services will host 3X more data [41]. As demand increases, carbon emissions from those services will grow. Already, the global IT sector emits more carbon dioxide than most countries [33]. Carbon emissions worldwide must decrease by 72% to prevent climate change [34]. The enormity of this challenge has prompted many IT companies to voluntarily reduce their emissions. If climate change continues unabated, all IT companies could be forced to reduce emissions if governments impose carbon taxes or if users boycott services with high emissions [6]. Carbon-aware approaches to manage data are needed.

Data replication is a widely used approach to manage data for Internet services. Replicating data across geo-diverse sites improves availability and reduces response times. Collectively, these sites are powered by many regional power authorities. Each authority manages the mix of resources to produce energy and thus controls local carbon emission rates (i.e., equivalent CO_2 released per joule). The energy mix varies from site to site because some regions may have a higher renewable energy supply. The energy mix also varies over time because renewable resources produce energy intermittently [24]. As a result, the carbon emission rate from such energy mix is fluctuating with the time and the location.

Prior work [26], [37] dynamically dispatches queries to the sites with lowest emission rates. This approach reduces a service's carbon footprints (i.e., its total carbon emissions), provided 1) the requested data is replicated to a low-emission site and 2) the query can be processed at any site [24], [26]. Figure 1(a) plots carbon footprints for a read-only query and a write query accessing the same data. The read query can be processed at any site and is dispatched to the site with lowest emissions. Replication decreases its footprint. The write query updates all sites that host the data. Replication increases its footprint. Finding a data replication policy (e.g., how many replicas and where to replicate) for low carbon footprints is challenging because:

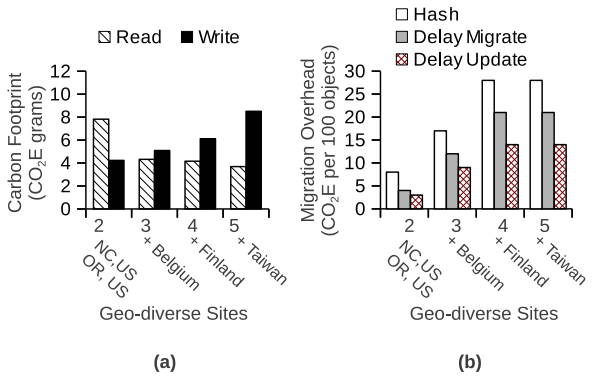


Fig. 1. (a) Querying data replication over Google's geo-diverse sites. Workload details are provided in Section VI-A. (b) Consistent hashing migrates data too often under time-varying heterogeneity.

1. Read-write mix, replication factor (i.e., number of replicas) and per-site emissions affect footprints. Replicating to too many or too few sites increases footprints.
2. Emission rates change over time. Replication policies based on outdated snapshots increase carbon footprints.
3. Data replication reduces effective storage capacity. Many services can not afford to replicate data to every site.

We present *CADRE, a carbon-aware data replication approach for geo-diverse services.* CADRE chooses the replication factor and replication sites for each data object. It supports workloads and time-varying emission rates and retains many features of consistent hashing, a widely used approach [20]. CADRE also provides high availability, respects system capacity limits and balances data across all sites.

CADRE builds upon prior work to forecast data energy needs and per-site emissions [38], [48]. We validate these approaches with traces from Google cluster [2], Yahoo! dis-

tributed file system, Yahoo! Answers [47], and World Cup [1]. CADRE combines these forecasts to generate *footprint-replication curves*, i.e., a model of each data object's carbon footprints as a function of replication factors. The footprint-replication curve exposes the best replication policy for each object. However, greedy per-object replication algorithms may violate global constraints on storage capacity or availability. CADRE uses an online, distributed algorithm to decide the replication policy based on the footprint-replication curve. Based on the multiple-choice secretary algorithm [22], our algorithm tries to replicate objects that benefit the most from the carbon-aware replication. We prototype CADRE to support frontend PostgreSQL applications [31] and evaluate it with emission data from the known locations of Google's data centers. Tests are conducted on physical machines (with direct energy measurements). We also use simulations for scale-out tests. CADRE provides a tradeoff between carbon emission savings and data availability, however, it does not necessarily lead to bad performance (i.e., latency). We empirically measure the carbon footprint and the per-query latency from using CADRE . Compared to consistent hashing, CADRE reduces carbon footprints by 70% while increasing the average latency by less than 5%. The performance difference between CADRE and the offline optimal algorithm is within 10%.

CADRE reduces carbon footprints when sites have spare storage capacity. Many services are unable to use all their capacity for many reasons, such as mixed workloads. Our empirical results show that with 30% spare capacity, CADRE uses 33% less carbon footprints than consistent hashing. When there is no spare capacity, CADRE performs as well as consistent hashing. CADRE also interacts well with latency- and cost-aware query dispatchers [32], [35], reducing footprints by 73% and 64%, respectively.

Our research contributions are as follows:

- An approach for data replication that extends consistent hashing and exploits time-varying emissions.
- A modeling approach that reduces a wide range of replication policies to functions that map replication factors to carbon footprints.
- An online algorithm that reduces carbon footprints for heavily accessed objects and adheres to capacity and availability constraints.
- A thorough evaluation with realistic workloads and emission traces that reveals up to 70% carbon savings.

The rest of this paper is organized as follows: In Section II, we describe consistent hashing and its shortfalls for carbon-aware replication. Section III outlines our approach and Section IV details its design. In Section V, we discuss our implementation concerns. We present results in Section VI and discuss our related work in Section VII. We conclude the paper in Section VIII and prove most of claims in the Appendix.

II. Motivation

Internet services strive to provide high availability and low response times. Replicating data to multiple sites improves availability. Fairly distributing workload across sites lowers response times. Consistent hashing, achieves both. The basic idea is to randomly assign data objects to both sites and backup sites. All data objects and sites are mapped to a circular ring. When an object is created, its unique name is used as a key to randomly assign the object to a contiguous range on the ring, called a partition. Partitions divide total storage evenly and each partition comprises storage from all sites. However, some sites may have more storage capacity than others. To account for heterogeneous sites, each site is randomly allotted virtual sites in proportion to its capacity. Virtual sites map randomly to regions on the circular ring. Replication is achieved by storing the object at the virtual sites adjacent on the ring (provided they do not map to the same site). Thus, the following properties hold: 1) data is randomly and fairly spread across sites, 2) if data access rates are independent of the naming scheme then workload is fairly distributed and 3) workload caused by recovering from failures is also fairly distributed.

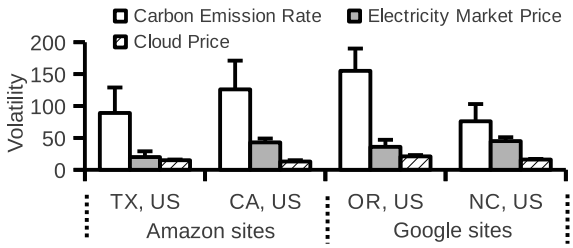


Fig. 2. Emission rates are more volatile than electricity and cloud prices. Volatility is the number of times the metric changes by more than 10% in a year. Error bars show the number of 5% changes.

The data service at each site is powered by local power authorities, which manages multiple energy sources, such as fossil-fuel power plants, solar farms and wind farms. Each joule consumed by data services is a mixed energy product from the dirty and the clean sources. The carbon emission rate m is defined as follows:

$$m = \sum_n \alpha_i E_i / \sum_n E_i \quad (1)$$

where α are the carbon dioxide emissions coefficient, E is the energy supply from one type of energy source (e.g., dirty energy from the fossil-fuel combustion), and n is the total number of energy source types available at one power authority. The intermittent energy supply from renewable sources leads to a different energy mix at one given time. As a result, the carbon emission rates, differ from classic storage heterogeneity, vary over time. Consequently, the local carbon footprints for a data object also vary. This section outlines two challenges presented by time-varying heterogeneity.

Carbon footprints caused by data migration: Consistent hashing supports adding and removing virtual sites on the fly. It migrates data to the next virtual site on the outer ring. If virtual sites are inversely proportional to emission rates, low-emission sites will host more data than sites with high emission rates. However, data migration increases carbon footprints by transferring data away from high emissions sites. Figure 1(b)

plots carbon footprints caused by migration under consistent hashing. It uses hourly emissions from regions where Google's data centers are located. It plots migrations for 1M replicated objects over 1 day as the number of sites increase. Sites are added by the distance from North Carolina. *Hash* uses the consistent hashing. Carbon footprints caused by data migration increase by 4X when the number of sites grows 2.5X.

Figure 1(b) also studies approaches to reduce migrations while using the same mechanism. First, *Delay Migrate* migrates data only if the amount of virtual sites for a site changes by more than 10%. This reduces migrations but hurts availability. It reduces aggregate migration footprints by up to 13% but does not reduce the growth rate as the number of sites increases. *Delay Update* updates emission rates after 8 hours instead of hourly. It smooths out jitters in the emission rate but adapts to emission rates less often, reducing its potential savings. This reduces migration overhead by almost 50% but the overhead still grows quickly as geo-diverse sites are added.

Figure 2 shows the root cause of data migration: *Per-site carbon emission rates change frequently.* Over one year, hourly emission rates in Oregon changed by more than 10% about 155 times and 5% up to 196 times [5]. When we sort these 4 sites by their emission rates, the ordering changed hourly. We chose these sites because their EC2 and electricity-market prices were also available [9], [14]. Compared to the storage prices on EC2, emission rates changed 7X more frequently. Dynamic prices in local electricity markets also change often, but less frequently than emission rates.

Using virtual sites to handle heterogeneity caused by volatile emission rates is similar to churn in distributed hash tables. Prior work used periodic and lazy migration policies to smooth migration workloads [10]. These techniques prevent network bandwidth saturation but do not reduce carbon footprints. Recent work constrains migration to adjacent nodes, which improves LAN performance, but in the geo-distributed setting, migration still incurs high carbon footprint [19].

Time-varying emissions reduce footprints for read-only queries but not writes: We target systems where clients interact with query dispatchers. Dispatchers lookup replication policies from a central source.

Our key insight is that data paths for read and write queries differ. Within a geo-diverse site, reads often use fewer hardware resources than writes, e.g., eschewing disks. Across sites, writes often require processing at many sites. Normally, reads are processed at 1 site but writes must (eventually) touch a majority. As a result, carbon footprints of read and write queries differ. Figure 1(a) plots these footprints as the replication factor increases under a model where writes are sent to all replicas. Replication reduces the footprints for reads but increases the footprints for writes. The best replication policy for each object depends on the mix of reads and writes.

III. CADRE Overview

Section II shows that consistent hashing can not exploit diverse emission rates at each site, because rates change too often. Time-varying rates could trigger too many data migrations, increasing carbon footprints. Further, the best replication factor depends on the mix of read and write queries, but consistent hashing only applies a static replication factor. Figure 3 outlines CADRE. It forecasts carbon footprints for each object and crafts policies that consider carbon footprints, load balance, storage capacity and availability. When a data object is created, CADRE stages it (i.e., temporarily assigns the object to sites). Then, it profiles the object's workloads and per-site emissions to forecast the object's carbon footprints across many replication policies. Finally, it assigns a carbon-aware policy for each object. Infrequently accessed objects are archived to make space for new objects. CADRE assumes there is enough storage capacity to support consistent hashing replication. CADRE policies have the following features:

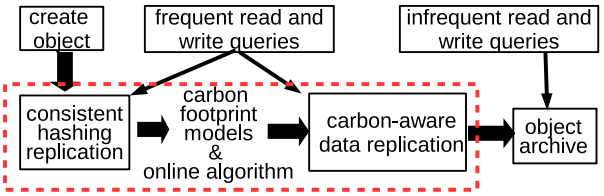


Fig. 3. Our replication approach, CADRE, extends consistent hashing with carbon footprint modeling and online replication algorithms.

Low-emission sites host data with heavy workloads: Objects with heavy access rates have large carbon footprints. These objects are replicated to sites that have low emissions during heavy workload periods. CADRE identifies and replicates objects with heavy access rates via staging.

Infrequent data migration: CADRE permanently places objects after profiling their workloads. It migrates each replica at most once. We used the setup in Figure 1(b) to study worst-case migration overhead in CADRE. The footprint is about 2.9 grams per 100 objects, a 68.7% reduction compared to the delay migration approach (e.g., Delay Migrate in Figure 1). More importantly, in CADRE, migration overhead stays the same as geo-diverse sites are added.

Objects are fairly spread across geo-diverse sites: CADRE uses two mechanisms to balance data across sites. First, it uses consistent hashing to determine where objects can be placed. It restricts replication to the sites that consistent hashing would use under a large replication factor. This approach alleviates imbalance because preferred, low-emission sites are allowed to host only a fraction of all objects. Second, CADRE supports per-site weights to balance placement across sites, allowing managers to override carbon-aware policies.

Each object is highly available and storage capacity is respected: CADRE replicates objects to multiple geo-diverse sites, providing the same availability as consistent hashing. It may replicate objects with heavy workloads to more sites to exploit varying emission rates. However, over-replicating objects can exhaust the storage capacity. CADRE's online algorithm reigns in over-replication by reserving the technique for only objects with the heaviest workloads.

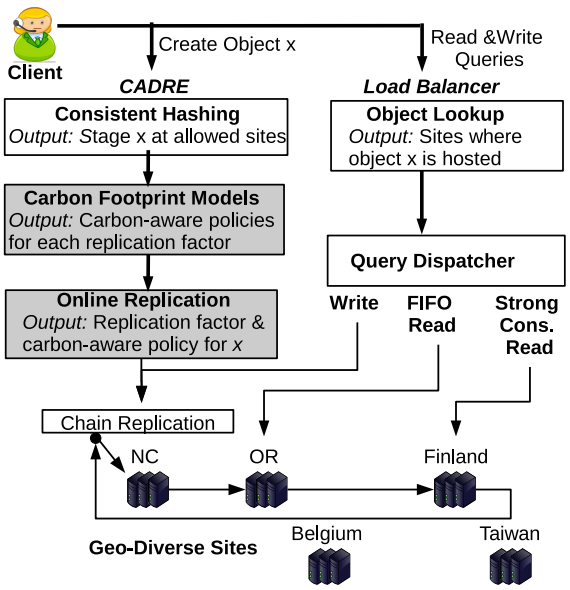


Fig. 4. Data paths for queries in CADRE. Boxes represent the system software that runs at all sites. Shaded boxes reflect CADRE components with novel design. CADRE assumes create queries precede read and write queries.

IV. CADRE Design

Figure 4 depicts the data path for create, read and write queries in CADRE. First, Section IV-A describes the system software that CADRE builds upon, i.e., the unshaded boxes in Figure 4. Specifically, create queries use consistent hashing and chain replication. Read and write queries use object lookup via consistent hashing and query dispatchers. Section IV-B details our carbon footprint models and Section IV-C details our online replication algorithm.

A. Architecture

Consistent hashing controls which sites are allowed to host an object. Given a maximum replication factor, only sites that would be selected by consistent hashing are allowed. This design provides a default write order for each object. Specifically, the order chosen by consistent hashing is also the order that sites process data updates. The order is preserved when carbon-aware policies choose a subset of sites for replication. The first site that hosts an object is called the head site. The last is called the tail site.

Create and write queries follow the chain replication protocol [40]. For each object, writes traverse geo-diverse sites in the same order. The head site updates first and sends the write to the next site. The write is complete when the tail site updates. Like CRAQ, each site keeps the most recent version for each object [39]. Recall, virtual sites randomly permute the order of physical sites across partitions. This mechanism increases throughput for durable writes that return after the completion from the first few, randomly ordered sites [8].

CADRE manages two *editions* of each object. The first edition means that the object has been replicated to its staging policy (i.e., the default replication policy from consistent hashing) and its workload is being profiled. The second

Symbol(s)	Meaning
Ω , Ω_j	Set of all sites and subset allowed for object j
O	Set of objects created
Q	Set of query dispatchers
e_r , e_w	Average energy per read and per write
r_j , w_j	Read/write queries for object j
$z_{i,q}$	Energy to message site i from dispatcher q
$m_i^{(t)}$	Emission rate at site i at time t
$C_j(D_j)$	Total carbon footprints for object j
K, k_j	Default replication factor and assigned factor for object j
$D_j^*(k)$	Sites that host k replicas for object j with min. footprints
$C_j^*(k)$	Minimum carbon footprints for object j with k replicas
k_j^*	Best replication factor for object j
$\mathbf{k}^o$	Best data replication policy
$C(\mathbf{k})$	Minimum footprint of data replication policy $\mathbf{k}$

TABLE I
SYMBOLS AND NOTATIONS USED IN THE MODELING PROCESS. **Bold** INDICATES VECTORS.

edition begins after the staging phase completes when an object has been replicated to its carbon-aware policy. For correctness, CADRE must ensure that writes and reads are properly directed to the right edition. When the staged edition is active, an object is replicated to the first K sites provided by consistent hashing, where K is the default replication factor. CADRE deactivates the staged edition before it activates the carbon-aware edition to ensure that outstanding writes are not lost. Specifically, it overwrites the staged edition with a forwarding message. New write queries are queued until the carbon-aware edition is activated. Read queries either return stale staged-edition data or they are queued.

In addition to each object's data, CADRE also stores replication policies for each object. Object lookup retrieves this policy. On a lookup, the dispatcher first checks the cache for replication policy. A cache miss causes the dispatcher to lookup the object using the hash function in consistent hashing, i.e., it indicates that the staged edition is active. When the dispatcher receives a signal that the staged edition has been deactivated, it gets and caches the carbon-aware policy.

CADRE allows virtual sites to leave and return. Power cycles and network issues affecting individual servers cause outages. Virtual sites recover lost objects when they receive a read or write query. On reads, the site contacts nearby virtual sites to restore lost data. On writes, the virtual site gets outstanding queries beginning from the last confirmed write from its predecessor in the write ordering. CADRE assumes physical sites are stable and virtual sites are assigned to sites over long periods.

Support for Eventual and FIFO consistency: Query dispatchers route queries between sites. To lower carbon footprints, for example, dispatchers should route queries to sites with low emissions. Chain replication allows for routing read-only queries; writes must update all replication sites. However, as discussed in prior work, routing policies affect consistency [39], [8]. A query dispatcher that randomly routes reads to any site can ensure clients that they will eventually (after many reads) retrieve the current version. Clients should compare version numbers of their recent reads to make sure a recent read has not returned an older value. As a result, this achieves the eventual consistency. A query dispatcher that

routes queries to the same (possibly randomly selected) site ensures clients that they will only read updates. Each client's writes are seen in order, i.e., FIFO consistency. CADRE supports either of the two access models. Although not required, we assume that each replication site hosts only one replica in the remainder of the paper.

By default, FIFO consistency forces dispatchers to route queries for an object to one site. However, dispatchers may need to change their routing policies when emission rates change. Fortunately, the policies change infrequently relative to read access rates, e.g., hourly to sub-second time scales. To ensure FIFO consistency, a dispatcher caches and returns the current versions of objects hosted at the old site. Periodically, they check the version at the tail site. When the versions match, the dispatcher routes subsequent queries to any site. CADRE implements this method to maintain FIFO consistency while dispatching queries to low emission sites.

B. Carbon Footprint Models

The carbon footprint for an object is the product of two variables: energy used to access the object and emissions rate during those accesses. CADRE uses an analytic model to describe how the energy coefficient changes across workloads and replication policies in one time frame $[T_1, T_2]$. The model is shown in Eq. (2), (3), and (4).

$$C_{j,r}(D_j) \triangleq \sum_{t=T_1}^{T_2} \sum_{q \in Q} (e_r + z_{i,q}^{(t)}) r_{j,q}^{(t)} \min_{i \in D_j}(m_i^{(t)}) \tag{2}$$

$$C_{j,w}(D_j) \triangleq e_w \sum_{t=T_1}^{T_2} w_j^{(t)} \sum_{i \in D_j} m_i^{(t)} \tag{3}$$

$$C_j(D_j) \triangleq C_{j,r}(D_j) + C_{j,w}(D_j) \tag{4}$$

Note, $C_{j,r}(D_j)$ and $C_{j,w}(D_j)$ are the total carbon footprints of reads and writes for object j, respectively. $z_{i,q}$ accounts for the lowest energy consumption of routing the read. All notations are defined in Table I. The models use average emission and access rates over discrete intervals. Section V details our approach to forecast these coefficients. Here, we study the model under perfect forecasts. First, the energy used at a single site for read queries differs from write queries. Write queries often access more resources than reads, e.g., hard disks. However, read-only queries often involve complex joins and scans [46]. For read-only queries, we also model communication cost between dispatchers host sites. However, prior work has shown that the energy footprints for communications are often the second-order effects compared to processing footprints [23].

To capture energy needs at a single site, we run benchmark workloads on a small cluster. The benchmarks execute representative read-only or write queries in succession, achieving high utilization. We measure the energy consumption and use it to estimate R/W energy needs. We use our R/W needs to model the total energy needs for TPC-H workloads in Postgres. In Figure 5, We compare our approach to 1) Per-request, a per-request cost model that does not distinguish

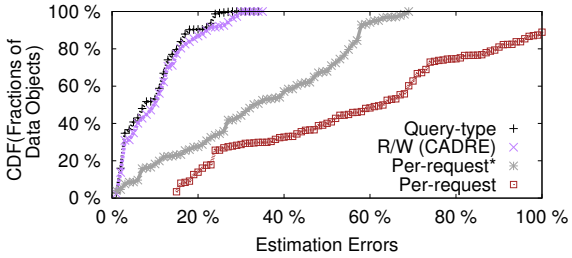


Fig. 5. CDF of modeling performance across five sites. *: best performance of the per-request model on one site.

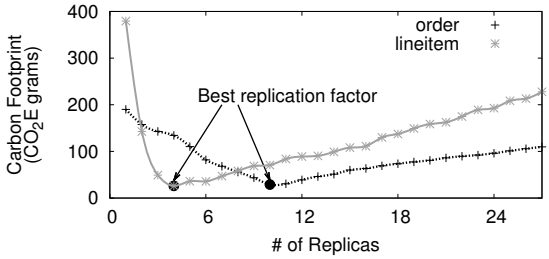


Fig. 6. The convex footprint-Replication curves.

reads from writes [26], [43] and 2) Query-type, a more advanced approach that further distinguishes read types, e.g., join, select and scan [46]. Results show that our approach provides a nice tradeoff between simplicity and accuracy, achieving 35% reduction in average error compared to the first approach while yielding only 10% more error variations than the second approach.

Creating footprint-replication curves: Replication helps dispatchers reduce footprints on read accesses, but it increases the footprints of write accesses. Using Eq. (4), we explore an object's total carbon footprints under all possible replication factors. The graph, where each point on the y-axis is the smallest carbon footprint achievable under the corresponding replication factor on the x-axis, is *a footprint-replication curve*. More precisely, $\mathbf{R}_j$: $R_{j,k} = C_j^*(k)$.

$$D_j^*(k) \triangleq \underset{D \subseteq \Omega_j, |D|=k}{\arg\min}\ C_j(D) \tag{5}$$

$$D_j^*(k-1) \subset D_j^*(k) \tag{6}$$

$$k_j^* = \underset{k=1,\ldots,|\Omega|}{\arg\min}\ D_j^*(k) \tag{7}$$

$$C_j^*(k) \triangleq C_j(D_j^*(k)) \tag{8}$$

Eq. (5), (6), (7), and (8) present optimization models for the footprint-replication curve. Eq. (6) ensures that if the replication factor decreases, e.g., a virtual site fails, the remaining virtual sites are unaffected. It requires that the best replication policies build upon each other. Without this constraint, a decrease in the replication factor could force CADRE to choose sites combined that lead to higher carbon footprints. Consistent hashing has this feature, called smoothness [20]. Given the inputs to our model, we construct best replication policy with a factor of k by incrementally adding a site from the best $k-1$ policy.

Footprint-replication curves can be divided into two parts. In the first part, carbon footprints decrease monotonically because replication to more sites provides more cost saving potential for reads. In the second part, carbon footprints increase because the routing provides little savings and writes dominate. The end of the first part is the best replication factor, because the footprint-replication curve is convex (Proofs are in the Appendix).

Figure 6 gives an example of two footprint-replication curves (i.e., lineitem and order data tables in TPC-H workloads [4]). The order table is a frequently accessed small table (100Qps+25K rows) and the lineitem table is a less popular larger table (1Qps+1M rows). Figure 6 shows that curves decrease with more replications added for read flexibility. However, the curve reaches the minimum point, where savings benefited from reads are compensated by penalties from keeping consistency for writes. After the point, writes' costs dominate the total carbon footprint and the curve is increasing.

C. Online Algorithms

Footprint-replication curves provide the best carbon-aware policy for each object. However, applying the best policy for each object may violate global system constraints. CADRE allows system managers to set the following constraints:

1. Storage Capacity: We target fast but costly in-memory stores that are widely used to provide low response times and high quality results. As shown in Eq. (9), managers can set the *provisioning factor* f to force partial replication because full replication is often too costly.

$$\sum_{|O|} k_j \leq |O||\Omega| f \tag{9}$$

2. Availability: Replication to geo-diverse sites ensure that objects are durable and available during earthquakes, fires and other regional outages. Eq. (10) ensures that every object is replicated to at least K sites but no more than $|\Omega_j|$.

$$K \leq k_j \leq |\Omega_j|, \forall j \leq |O| \tag{10}$$

3. Load Balancing: Consistent hashing uses virtual sites to handle heterogeneity [12]. CADRE overrides these settings to consider time varying emission rates. Managers can devalue emission rates by setting per-site weights, shown in Eq. (11). The weight vector can be changed on the fly.

$$m_i^{(t)} \triangleq m_i^{*(t)} w_i. \tag{11}$$

CADRE finds good replication policies that 1) respect these constraints and 2) reduce carbon footprints. Next, we first present a greedy algorithm that assigns each incoming object to its best replication factor until the storage capacity is exhausted. From this algorithm, we devise an optimal offline algorithm. After studying the shortcomings of the greedy algorithm, we propose an approach based on the multiple-choice secretary algorithm.

Greedy online algorithm: The greedy algorithm proceeds as follows. When a data object is created, we compute its optimal replication factor using its footprint-replication curve. At the start, we always replicate the object to its best replication factor unless the availability constraint is violated. However, if the best replication factor would cause the spare capacity to fall below the minimum capacity required for the remaining objects, we only replicate the object to K sites (i.e., the minimum replica for availability). At the end of this greedy algorithm, all objects are replicated to either their optimal policy or the default policy. Algorithm 1 shows the pseudo-code of the Naive online algorithm.

Algorithm 1 Naive online algorithm

Ensure: $\forall j = 1, \ldots, |O|, k_j \leq k_j^*$ or $k_j = K$

```
C_remain := |O||Ω|f
k = (k_1, ..., k_|O|)^T := 0
for j := 1 TO |O| do
    if k_j^* ≤ K then
        C_remain := C_remain − K
        k_j := K
    else if C_remain − k_j^* > (|O| − j)K − K then
        C_remain := C_remain − k_j^* {Storing k_j^* replicas will still guarantee enough space for the remaining objects. We will store k_j^* replicas.}
        k_j := k_j^*
    else
        C_remain := C_remain − K {Storing k_j^* replicas of the object will not allow us to store at least one replica for the remaining objects. In this case, we only store one replica for the jth object.}
        k_j := K
    end if
end for
return k
```

If there always exist spare capacities for data replication, the greedy algorithm is already optimal (Proofs are in the Appendix). However, in the worst case, all late arriving objects with heavy workloads could be assigned to default replication policies. We propose, Fill-and-Swap, an offline algorithm that finds an optimal solution.

Fill-and-Swap algorithm: Given the result from the greedy algorithm, we use gradient search to find a local optimum. First, we *fill* the unused storage capacity by increasing replication factors of objects that are not replicated to their k_j^* sites. Specifically, we increase the replication factor for one at a time, choosing the object that can reduce the most global carbon footprint. Once the storage capacity is fulfilled, we further reduce the footprints by *Swap*. adding replicas for some object who has less replicas than its k_j^*; and In the *Swap*, We find the ith object that reduces the most carbon footprints if its replication factor is increased by one; and find the jth data object that increases the least carbon footprints if its replication factor is decreased by one. If *Swap* reduces the global carbon footprints, we keep looking for this kind of object pair (i, j) and perform *Swap*, otherwise, the algorithm terminates.

This gradient search approach yields the optimal solution because footprint-replication curves are convex. However, it

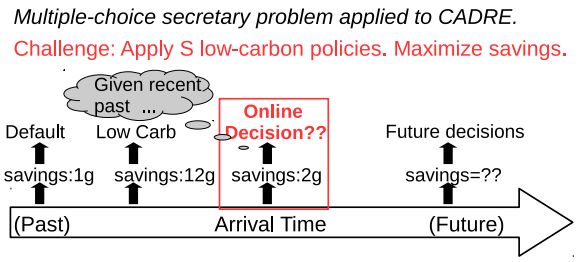


Fig. 7. Online replication within CADRE uses the multiple choice secretary algorithm [22].

only works offline by scouring footprint-replication curves of all objects. Note, replications where $K > k_j^*$ indicate that the minimum number of copies for high availability is already sub-optimal. K is already the best setting for these objects thus they are safely ignored in this discussion.

Competitive Bound Analysis: The competitive bound normalized to $|O|$ of the greedy online algorithm is $1+(N-\alpha\frac{C}{M})\theta$. where θ is a trace dependent constant. Importantly, Fill-and-swap resolves two problems with the greedy approach. First, spare capacity is exhausted too quickly. Late arriving objects are forced to accept the default policy. Second, some objects could have small total benefits but relatively large gradients at small replication factors.

Next, we derive the greedy-to-offline competitive ratio to explore the theoretical performance bound. Let l_1 and l_2 to denote the number of objects that the greedy approach assigns to best and default policies, respectively. In the worst case, $|\Omega|l_1+Kl_2=|O||\Omega|f$, meaning the first l_1 objects are fully replicated. By the definition, $l_1+l_2=|O|$, thus, we get $l_2=\frac{|O||\Omega|(1-f)}{|\Omega|-K}$. Also, in the worst case, every object in l_2 should have been replicated to achieve the maximum carbon savings instead of those in l_1. Finally, we define the maximum carbon footprint savings from one single swap is β. Given β, we know that the maximum additional footprint introduced by the greedy approach is $l_2(\Omega-1)\beta$. This leads to a competitive bound of $1+|\Omega|(1-f)F(\beta)$, which, unfortunately, grows with the number of sites.

Multiple-choice Secretary algorithm: Empirically, it is likely that late arriving but frequently accessed objects account for most of the carbon difference between greedy and offline algorithms. This result makes sense given that data access patterns often follow top-heavy Pareto or Exponential distributions [38], [48]. The challenge is to judiciously select the objects with largest carbon footprint savings when they arrive, using only knowledge of previous objects.

CADRE uses the multiple-choice secretary algorithm, shown in Algorithm 2, to select l_1 objects with largest savings [22]. Given a stream of numbers, the challenge is to select l_1 numbers that have maximum sum. If $l_1=(O_u-O_l)$, the algorithm already picks one of the largest object j to replicate optimally (Line 5-6). If $l_1>1$, the algorithm randomly samples a binomial distribution y=B(l_1;$\frac{1}{2}$) (Line 7-12). It then recursively observes j objects from the half of the remaining objects

Algorithm 2 MCSA for Carbon-Aware Replication

1: **Function** MCSA(l_1, O_l, O_u);
2: l_1 := min($\frac{|O|(|\Omega|f-K)}{|\Omega|-K}$, l_1);
3: O_u := min($|O|$,O_u);
4: O_l := max(0,O_l);
5: **if** $l_1>(O_u-O_l)$ **then**
6: **return** $k_j:=k_j^*$;
7: **else if** $l_1>1$ **then**
8: **if** $j\le floor(\frac{O_u-O_l}{2})$ **then**
9: **return** MCSA($B(l_1;\frac{1}{2})$, O_l, $O_u:=floor(\frac{O_u-O_l}{2})$);
10: **else**
11: **return** MCSA($B(l_1;\frac{1}{2})$, $O_l:=ceil(\frac{O_u-O_l}{2})$, O_u);
12: **end if**
13: **end if**
14: **if** $k_j^*\le K$ **or** $j\le O_l+\frac{O_u-O_l}{e}$) **then**
15: **return** $k_j:=K$;
16: **end if**
17: $Largest := max(C^*_{O_l}(K)-C^*_{O_l}(k^*_{O_l}), C^*_{O_l+\frac{O_u-O_l}{e}}(K)-C^*_{O_l+\frac{O_u-O_l}{e}}(k^*_{O_l+\frac{O_u-O_l}{e}})$;
18: $Obs:=O_l$;
19: **while** $Obs\le j-1$ **do**
20: $Savings_{Obs}:=C^*_{Obs}(K)-C^*_{Obs}(k^*_{Obs})$;
21: $Obs:=Obs+1$;
22: **if** $Savings_{Obs}\ge Largest$ **then**
23: **return** $k_j:=K$;
24: **end if**
25: **end while**
26: **if** $Savings_j>Largest$ **then**
27: $k_j:=k_j^*$;
28: **else**
29: $k_j:=K$;
30: **end if**

and record the largest savings (Line 18-23). Any objects that has a larger saving afterwards will be picked (Line 25-29). After l_1 objects are selected, the algorithm terminates. The algorithm guarantees that the sum of carbon footprint savings from selected objects is within $O(1+\frac{5}{\sqrt{l_1}})$ [22]. Note, l_1 can be also represented as a function of the remaining capacity. Given modern storage systems stores millions of objects, we expect l_1 to be a large number, meaning the algorithm is likely to find the objects that would lead to maximum savings.

V. Implementation

Section IV outlined a system design that supports eventual and FIFO consistency, carbon footprints models and online data replication. Inaccurate forecasting degrades online replication, increasing footprints. This section outlines key insights that improve forecasts.

Access rates decay quickly then slowly over time: Prior research has shown that per-object access rates fit Exponential and Pareto distributions well [38], [48]. Also, it is now common for online services track read and write accesses per object. We use historical traces to determine which functional

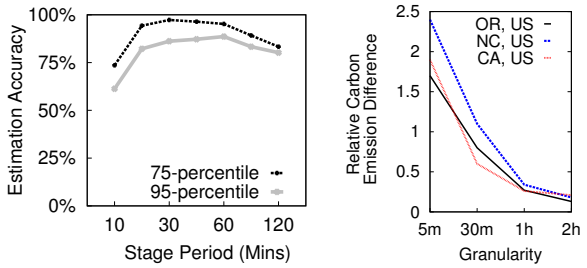


Fig. 8. (a) The effect of staging duration on estimation accuracy in Yahoo! System trace. (b) Emissions forecasts improve under coarse granularity.

forms provide low forecasting error. Specifically, we measure goodness of fit via the Kolmogorov-Smirnov (KS) test [11]. KS tests find the maximum discrepancy between an empirical and the theoretical distributions, in this case, observed hourly R/W access rates and Pareto modeled rates. We perform KS tests to all data traces in our experiment. Results shows about 83.3%, 97.8% and 74.7% of data objects, from the Yahoo! System trace, the WorldCup trace and the Google Cluster trace, respectively, have KS statistic converging towards zero.

When we plotted raw accesses, a clear pattern emerged: Shortly after an object is created, it achieves its highest access rates. Then, it becomes old and its access rates drop precipitously. Finally, there is a long period in which objects are accessed rarely. This period is much longer than the period when data are popular. From CADRE's perspective, this long period is likely to be the end of an object's lifetime. Similar patterns can also be found by analyzing popular social media data (i.e., tweets and Facebook updates) and small files in the Hadoop file system [21].

Staging periods improve accuracy significantly: Only a few observations are needed to calibrate Pareto distributions. It reduces the amount of data that needs to be collected during the staging phase. However, these observations are crucial for accurate forecasts. They explain the magnitude of an object's popularity, its decay rate and likely lifetime. We compare the estimation accuracy of different staging durations in the Yahoo! File System trace, as in Figure 8(a). A 30-minute staging period yields average error below 7% (i.e., above 93% accuracy) for 75% data objects. A 30-minutes period represents about 2% of an average lifetime of data objects.

Forecasting at coarse granularity: Recall, carbon emission rates vary across sites and over time. They are set by regional power authorities, not service providers. They are often in flux, because of the intermittent energy supply. On one hand, solar panels and wind turbines can reduce the load on dirty sources, lowering emission rates. Unfortunately, these sources have unpredictable outages (e.g., cloud covers solar panels) [37]. On the other hand, regional authorities often use dirty sources when they need on-demand energy production to balance energy demand surges, raising regional emission rates.

Emission rates are easier to predict at coarse granularity. We obtain carbon emission rates using models from the National Renewable Energy Laboratory (NREL) Wind Integration Dataset [5]. The emission rate at time t is the weighted average of emissions for the site's region and the contribution of a regional wind farm. We implemented regression models recommended by the US Environmental Protection Agency (EPA) to forecast emission rates [3]. These models use weather information based on the last 2 hours and estimated carbon data based on history. Similar models have also been used in systems research for per-site carbon emission forecasts [17].

Figure 8(b) shows the worst-case relative carbon emission forecasts at different granularity. The relative difference is calculated by $|\frac{\text{Estimation - Measurement}}{\text{Measurement}}|$. We observe up to 3X error when forecasting only minutes ahead. It is due to short term issues like intermittent outages. The worst-case error falls to 1.25X when forecasting over 1h granularity.

VI. Evaluation

A. Experiment Setup

Our prototype has three parts. First, the core software implements functions of consistent hashing, carbon footprint models and replication algorithms. These packages run at each site. Second, a front-end PostgreSQL database engine handles workloads on a specific site. The engine is also modified to include a background process that periodically computes local emission rates. Finally, the query dispatcher routes queries between sites. Note that, in our current prototype, we ignore the network cost between sites and end users, consider it as a future work in implementation.

We study our prototype in two environments. First, we have a real testbed comprising four clusters, which represent four Google sites (i.e., NC, SC, IA, and OR) in the United States. Each cluster has a location-based carbon emission profile, and a server profile similar as the machine used in Google sites. There are four DELL R710 servers per cluster. Each server has 12 2.4Ghz cores, 16 GB RAM and 1 TB disk-based storage. DVFS and other power saving mechanisms are turned off to measure energy in the worst case. This provides references for e_r and e_w in Eq. (2) and (3). To study CADRE at scale, we also set up simulations. Starting from the site profiles from our physical testbed, we keep adding new sites based on the known profiles of Google and Amazon sites [29], [18].

Traces and Workloads: For each data object, we consider three types of queries, namely create, read and write. In the test, we treat a data table as one data object. We evaluated CADRE with data access traces from *WorldCup*, and *Google*.

WorldCup is the three-month webpage access data during World Cup 1998, which covers over 1 billion queries [1]. This trace only contains two types of queries–create and read. This trace is representative of in-memory data accesses with simple and heterogeneous transactions having skewed access patterns. The Google trace is a one-month data access trace on a google cluster of over 11k machines in May 2011 [2]. This trace contains data access information such as access type (i.e., create/read/write), start/end time, etc. Using this trace, we can evaluate the estimation performance from the staging phase in CADRE. In our test, we select one-week data on 12 machines

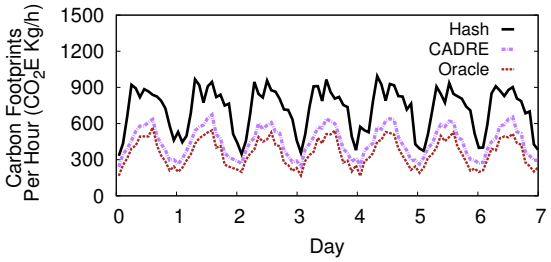


Fig. 9. Carbon footprints for a 1-week snapshot of Google trace. Emission and workload rates are provided hourly.

from the entire trace. The minimum R/W ratio per hour in the selected trace is 1.2:1. Note that, both WorldCup and Google traces are representing real-world data access patterns but not real data. To fill queries with real data, we use TPC-H, TPC-W and TPC-C benchmarks [4]. In our testbed, we have over 2,000 objects per site, corresponding to 600 GB of data.

Baselines: For evaluation, we implement the following data replication strategies:

- *Hash* implements consistent hashing shown in Figure 1(b) with a replication factor of 3.
- **CADRE** implements our full solution, including workload and emissions forecasts, and the online replication.
- *Blind* removes the staging period for forecasting access rates. It is based on the average access rates per hour across all objects.
- *Greedy* replaces MCSA with the greedy approach.
- *Oracle* replaces MCSA with Fill-and-Swap, i.e., the offline algorithm that knows perfect prediction, thus the staging phase is not needed.
- *Weighted* constrains CADRE by setting strict weights on the fly to ensure no site hosts more than its fair share (based on the capacity) for more than 8 hours.

Other than different data replication strategies, we also studies how CADRE interacts with routing policies:

- Random routing (*Rand*). It dispatches queries randomly to any site with available data. Rand is the default dispatching policy unless specified.
- Latency-aware scheduling (*LW*) [35] routes the request to the nearest site with available data.
- Cost-aware routing (*CW*) [28] dispatches the request to the site with lowest electricity cost.
- Green load balancing (*GLB*) [26] dispatches the request to the site while considering both cost and latency.

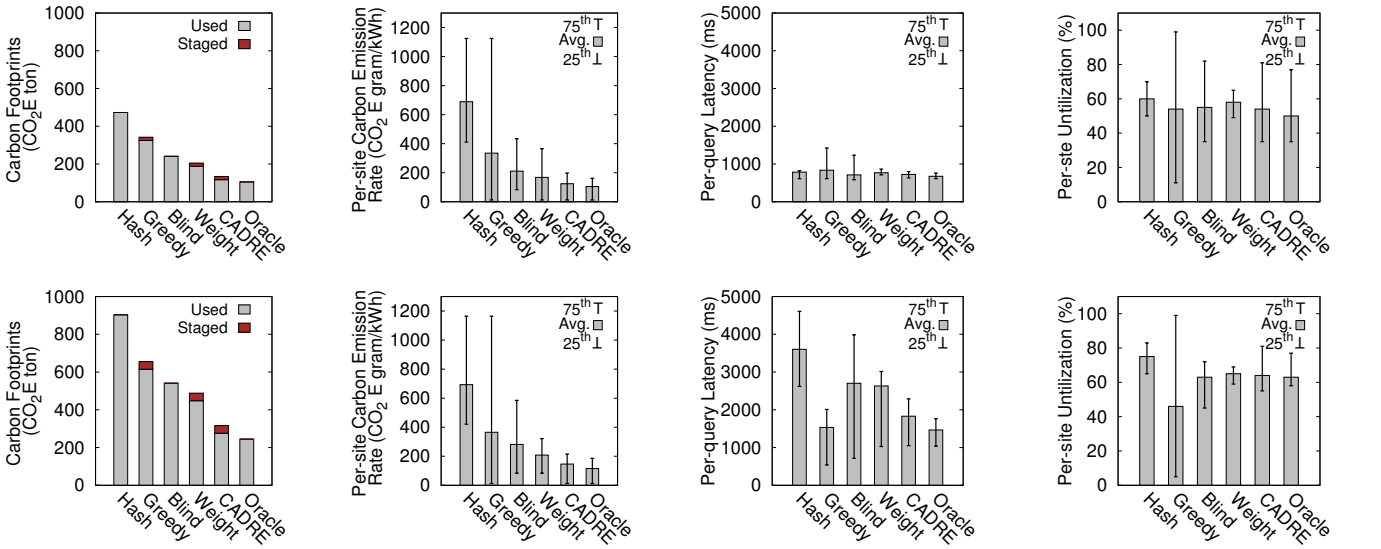
B. Experimental Results

Data services have much smaller carbon footprints under carbon-aware data replication: We start by showing a 1-week carbon footprint snapshot under consistent hashing and carbon-aware data replication strategies in Figure 9. All carbon footprint curves have diurnal patterns due to the diurnal workload patterns from the Google trace. Across the whole trace, CADRE achieves lower and more predictable carbon footprints. Its 25th and 75th percentiles differ by 25%, whereas these percentiles differ by 63% with *Hash*. Meanwhile, CADRE achieves the similar performance as *Oracle*. For over 90% data points, the carbon footprint difference between CADRE and *Oracle* is within 10%. Such difference consists of the carbon footprint overhead from the staging phase in CADRE, and prediction errors. Note that, we use Rand to route requests in this test. In the rest sections, we assume Rand is the default routing policy unless specified. In all, by making data replication carbon-aware, the carbon footprints of serving data are much smaller we can save a lot of carbon footprints from data services.

Performance comparison of different replication strategies: To examine the performance of CADRE, we compare it with other baselines using WorldCup and Google (the first and the second row in Figure 10, respectively). In the experiment, we measured the total carbon footprints, the statistics of carbon emission rate, the latency, and the per-site capacity utilization. Note that, the latency for writes is measured as the duration of all chain replications are updated after the write.

Figure 10(a) shows that CADRE outperforms *Hash*, beats other baselines, and performs closely to Oracle. The carbon footprint difference between CADRE and *Hash* in Google is 2.5X larger than that in WorldCup. It is because Google is a read/write mixed workload and CADRE considers both reads and writes, thus CADRE can provide a better tradeoff under this kind of workload. CADRE beats other carbon-aware replication strategies, such as *Greedy*, by saving up to 56% carbon footprints. In contrast to those baseline strategies, CADRE focuses the set of data objects that contribute the most to carbon savings, and respects both capacity and availability constraints. it still can save 38.3% carbon footprints due to our online replication algorithm. Thus, for some blackbox systems that the workload prediction is impossible, one can still use mechanism of CADRE to save carbon footprints. We can also observe the reasons behind the carbon footprint differences in Figure 10(a) from Figure 10(b). Except *Oracle*, CADRE has the least average carbon emission rate across all replication sites. When we examine the 25^{th} carbon emission rate data, CADRE always picks the cleanest site as *Oracle*, whereas *Blind* and *Weight* fail (in the Google trace) because of the misprediction and the strict load balance constraint. For the 75^{th} data, we observe that *Greedy* picks the most dirty site as *Hash*, which significantly reduces its carbon savings. It is because Greedy aggressively replicates optimal replication factor for the first-come data and quickly exhausts the capacity, thus it has to replicate more data in the dirty sites.

Using carbon-aware data replication does not necessarily hurt the performance of serving data. As illustrated in Figure 10(c), the average latency in CADRE is only 5% more than in *Hash*. Introducing writes increases the latency because chain replication requires queries to traverse every replication site. However, CADRE uses its model to choose a good combination of sites. We observe the latency decreases from Hash to *Blind* (no workload model) to *Weight* (constrained model) to CADRE. Note, *Greedy* exhausts spare capacity quickly and assigns late arriving objects to their minimum replica-



(a) Carbon Footprint (b) Carbon Emissions Rate (c) Latency (d) Average Capacity Utilization

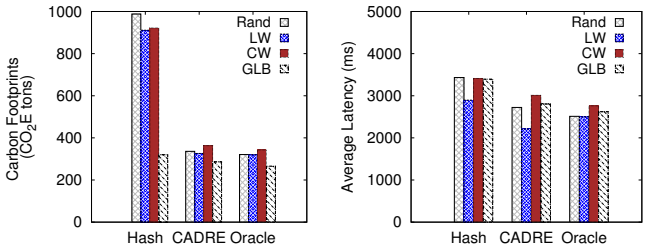
Fig. 10. Performance comparison using the WorldCup trace (First row) and the Google trace (Second row).

tion factor. This reduces the latency for writes but increases carbon footprints. The difference between 75-percentile data of CADRE and Hash is larger because, requests may route to clean sites that have long distance from the original sites. However, for Google, the average turnaround time of CADRE is much smaller than *Hash*, as CADRE usually picks less sites than Hash. The shorter length of the replication chain has, the less time needed for keeping consistency. As a result, CADRE achieves 23% less average turnaround time than *Hash*.

At last, we examine the per-site capacity usage in Figure 10(d). *Weight* has the similar variance (10%) in the capacity utilization as *Hash*, as both strategies are designed for the fair load balancing. Comparing to *Hash* and CADRE, *Weight* serves as a tradeoff between the load balancing requirement and the carbon footprint reduction. *Greedy* has the largest variance (46%) for both workloads, due to its aggressive replication behavior. In all results, CADRE has the closest performance to *Oracle*.

All results demonstrate that CADRE can save carbon footprints from serving data across geo-diverse site, even under a random routing policy. Clearly, a carbon-aware routing policy can further exploit the low carbon feature from CADRE.

CADRE interacts well with many routing policies: We here examine how CADRE interacts with routing policies for carbon reduction, in Figure 11. Although Rand, LW, and CW are not carbon-aware routing policies, compared to *Hash*, CADRE can still reduce up to 70%, 73%, and 64% of their carbon footprints, respectively (Figure 11(a)). It is because the routing policies are running under the carbon-aware replication from CADRE. Using the replication from CADRE, LW can further reduce 38% latency than that of *Hash*+Rand (Figure 11(b)). Using CADRE+GLB helps to save additional 21% carbon footprints with 17% less in average latency than that from Hash+GLB. Although we did not show the electricity cost in the figure, using CADRE's replication,



(a) Carbon Footprints (b) Latency

Fig. 11. (a) Carbon footprints and (b) the average latency comparison between replication strategies and routing policies.

CW has 23% less electricity cost than *Hash*+CW.

CADRE is designed to exploit time-varying emission rates. However, other important costs also vary over time, e.g., electricity prices. Often, these metrics change less frequently than emission rates, meaning that consistent hashing may perform better relative to CADRE. We did a study on the electricity price analysis from [24], the result shows, by simply placing data in synesthetic with the sites with low electricity price, CADRE can help saving 43% more electricity expenses, however, the relative carbon footprint increased. In the future, carbon emissions will better align with price, making CADRE an attractive approach for both metrics. Considering the price of renewables would decrease in the near future [42], the savings shall increasing with smart data replication decisions. Note that, our approach is orthogonal to the existing approaches that focus on workload and hardware [42], [32], [28].

Note that the analysis on the CW could be biased, because of the high market price of the renewable energy. As the development of sustainable energy and shrinking supply of coal, the renewable energy price would be offset and become cheaper in the near future. In this case, CADRE can help CW to save even more cost in electricity. In this case, the results of CW would converge to the result of GLB.

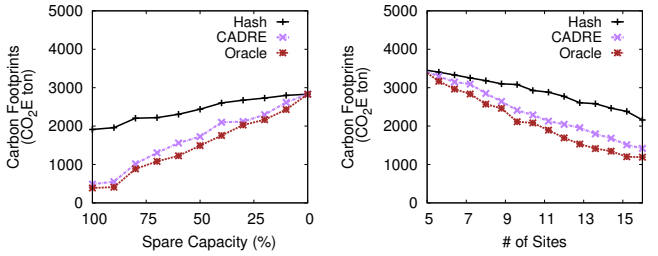


Fig. 12. Data replication simulations under (a) decreasing spare capacity and (b) scale-out.

From empirical results, we can see that CADRE serves as a good foundation for supporting carbon-aware routing policy designs. CADRE is an orthogonal effort to reduce the carbon emission, compared with the state-of-the-art carbon-aware routing policies. Combine them together can help achieve more carbon reduction in serving data across geo-diverse sites.

Spare storage capacity limits carbon footprint savings: We also evaluate the performance of *Hash*, CADRE and *Oracle* under bounded available capacities. The available spare capacity is gradually reduced from 100% to 0%, and carbon footprints of all data replication approaches are increasing, shown in Figure 12(a). The carbon footprint of *Hash* is increasing because more data migrations occur due to limited space. The performance of CADRE and *Oracle* are converged to *Hash* when there is no spare capacity. With 30% spare capacity, CADRE can still save up to 33.7% carbon footprints. In practice, although the storage and the computation capacity may be fully reserved by data services, most services cannot fully use all their resource reservations at the same time [25]. Thus, CADRE can help those services reduce their carbon footprints.

CADRE keeps low carbon footprints as more geo-diverse sites are added: We also study the scale-out performance of CADRE via simulations from 5 sites to 16 sites, using duplicated profiles of Google and Amazon sites, shown in Figure 12(b). Adding more sites with time-varying carbon emissions into the system, CADRE can further reduce carbon footprints of serving data (about 200 tons per site). Meanwhile, with the increasing number of sites, the performance of CADRE and Oracle are converging.

VII. Related Work

CADRE is a replication approach that considers time-varying emission rates. It uses spare capacity at geo-diverse sites to ensure that low-emission sites host carbon-heavy objects. The technical contributions are 1) an approach to model carbon footprints based on read and write data paths and 2) an online replication algorithm that respects capacity and availability limits. Below, we discuss related work.

Using geo-diverse sites to reduce costs: SPANStore replicates data to sites with low prices for storage and network transfer [44]. Like CADRE, it models the effect of replication, and finds optimal policies. However, CADRE considers emission rates, which change more often than storage prices. SPANStore migrates data when prices change significantly. As shown in Figure 1(c), carbon emissions change so often that migration approaches can increase carbon footprints. To reduce migration costs, CADRE profiles and forecasts access rates for each object and uses an online replication to make permanent, carbon-aware policies. To improve the accuracy of our forecasts, CADRE temporarily stages objects when they are created to collect early access rates. Shankaranarayana et al. consider the impact availability, which changes less often than prices [36]. CADRE compliments other environmental cost-aware approaches that route queries among geo-diverse sites [45], [26], [24]. These approaches often assume data is fully replicated and focus on read workloads. Using replications from CADRE, these approaches have significantly lower footprints than using replications from consistent hashing. CADRE also yields lower footprints for routing approaches that target other cost metrics, e.g., latency [35] or economic cost [32].

Novel replication approaches: Many recent works improve consistent hashing for geo-diverse services. Paiva et al. add locality aware placement for objects accessed together, which achieve 6X speedup [30]. ChainReaction improves concurrency by allowing write operations to complete before all replicas are updated [8].

CADRE places objects created at the same time at the same sites, reducing network latency and making placement policies easier to understand. Data paths for read and writes are affected by consistency settings. Transactional consistency is convenient for application programmers to use but challenging for database programmers to scale. Eventual consistency is easier to scale but inconvenient to use [12]. Causal consistency falls between these extremes, allowing programmers to reason about the order in which updates are applied while only passively exchanging data [27]. CADRE supports all those consistency settings using the chain replication in the system design.

Sustainable computing systems: Early work described research problems for renewable-powered data centers, including using load balancing to match power demand with supply [37]. Adding to this work, [13] studied the effect of low-level electrical equipment on renewable energy usage. Many recent systems have tried to implement geographic load balancing with real renewable powered systems. These papers often focus on the operating system and middleware tools needed to support such systems [45], [33], [49], [16], [15], [28]. Researchers have proposed to various approaches to maximize of renewable energy usage along the temporal dimension [7], [17]. In contrast, our work concentrates on the data management and related carbon emissions for a network of data centers. Although our prototype only contains a few clusters, they are representing different nodes at geo-diverse locations.

VIII. Conclusion and Future Work

Internet services replicate data across multiple geo-diverse sites. We have shown that consistent hashing inflates carbon footprints because: (1) frequently moving data away from sites

with high emission rates incurs high carbon footprint overhead, (2) objects with frequent reads are often under replicated, forcing dispatchers to serve data from sites with high emission rates and (3) objects with frequent writes are often over replicated, because writes must eventually propagate to high emission sites. CADRE replicates data to sites that combine together to yield low footprints for each object. Our contributions are (1) a modeling approach that considers data paths and the effect of replication for read and write queries and (2) an online algorithm that reduces the frequency of data migration. Our experimental results show that it is possible to reduce a significant amount of carbon footprints (70%) during data replication without sacrificing much data availability, latency, load balancing, and scale. Immediate future work includes adding the latency and the consistency constraints into the cost model, and expand the model with the network routing cost to the end users.

ACKNOWLEDGEMENT

We would like to thank our shepherd Dr. Diwakar Krishnamurthy for improving this manuscript.

APPENDIX

JOINT-ACCESS CONSTRAINT

Without losing generosity, we assume data objects are created at the same time and their access frequencies are linearly correlated. Such transactional groups are very common in practical systems. To reduce inter-site workload, objects within the same group should have at least one copy within the same site. To guarantee this property, we argued that joint-access constraint is sufficient. In this section, we will give the details about the proof.

Remember that joint-access constraint says:

$$D_j^*(k) \subset D_j^*(k+1). \forall j \in O, k < |\Omega|$$

Lemma A.1: If $\forall j \in O, k = 1, \ldots, |\Omega| - 1, D_j^*(k) \subset D_j^*(k+1)$, then $\forall j, p$ in transactional group O' either $D_j^*(k_j^o) \subseteq D_p^*(k_p^o)$ or $D_p^*(k_p^o) \subset D_j^*(k_j^o)$

Proof:

$$\begin{aligned}
&\because j, p \in O', \\
&\quad T_a(j) = T_a(p) \\
&\quad r_j^{(t)} = \lambda_{j,p} r_p^{(t)} \\
&\quad w_j^{(t)} = \lambda_{j,p} w_p^{(t)} \\
&\therefore \forall D \subseteq \Omega, C_j(D) = \lambda_{j,p} C_p(D) \\
&\therefore \forall k = 1, \ldots, |\Omega|, D_j^*(k) = D_p^*(k) \\
&\because \forall j \in O, k = 1, \ldots, |\Omega| - 1, D_j^*(k) \subset D_j^*(k+1). \\
&\therefore \forall j \in O, k < k' < |\Omega|, D_j^*(k) \subset D_j^*(k') \\
&\because \forall k = 1, \ldots, |\Omega|, D_j^*(k) = D_p^*(k) \text{ and} \\
&\quad \forall j \in O, k < k' < |\Omega|, D_j^*(k) \subset D_j^*(k') \\
&\therefore \text{If } k_j^o < k_p^o, D_j^*(k_j^o) \subset D_p^*(k_p^o). \\
&\quad \text{If } k_p^o < k_j^o, D_p^*(k_p^o) \subset D_j^*(k_j^o). \\
&\quad \text{If } k_p^o = k_j^o, D_p^*(k_p^o) = D_j^*(k_j^o).
\end{aligned}$$

■

PROOF OF CONVEXITY OF FOOTPRINT-REPLICATION CURVE

For all data object, their footprint-replication curves are convex under joint-access constraint. More precisely, $\forall j \in O, C_j^*(k)$ is a convex function if $\forall j \in O, k = 1, \ldots, |\Omega|, D_j^*(k) \subset D_j^*(k+1)$.

For the jth data object, we can order all datacenters according to the order by which the datacenter is selected to store the replicas for the data object. More precisely, the kth datacenter to the jth object is in the set $D_j^*(k) \setminus D_j^*(k-1)$, where $D_j^*(0) = \emptyset$. Since $D_j^*(k) \setminus D_j^*(k-1)$ contains exactly one element, then the kth datacenter to the jth data object is uniquely defined. We use $M_{j,k}^{(t)}$ to refer to the minimal unit price among the first k datacenters at time t. This means $M_{j,k}^{(t)} = \min_{i \in D_j^*(k)}(m_i^{(t)})$.

Let $T = \{T_1, \ldots, T_2\}$, which is the set of all times we consider. By sorting all datacenters against the jth data object, we denote the carbon ration of the kth datacenter at time t as $m_k^{(t)}$. Given the jth object and its kth datacenter, we define $T_{j,k} = \{t | t \in T, m_k^{(t)} \leq M_{j,k-1}^{(t)}\}$. By definition, $T_{j,k}$ contains times at which the kth datacenter has the minimal carbon ratio among the first k datacenters. Since we always choose the datacenter with least carbon ratio to read, if the jth data object stores its replicas in the first k datacenters, then the read operations would happen in the kth datacenter at $t \in T_{j,k}$.

Let $\Delta C_j^*(k) = C_j^*(k-1) - C_j^*(k)$. If $k \leq k_j^*$, then $\Delta C_j^*(k) \geq 0$. According to the definition of $C_j^*(k)$, we have

$$\Delta C_j^*(k) = e_r \sum_{t \in T_{j,k}} r_j^{(t)} (m_k^{(t)} - M_{j,k-1}^{(t)}) - e_w \sum_{t \in T} w_j^{(t)} m_k^{(t)}$$

Lemma A.2: For all $j \in O$ and $k \leq k_j^*$, if $D_j^*(k) \subset D_j^*(k+1)$, where $k = 1, \ldots, |\Omega| - 1$, then $C_j(\tilde{D}_j^*(k)) \leq C_j^*(k-1) - \Delta C_j^*(k+1)$ where $\tilde{D}_j^*(k) = D_j^*(k-1) \cup \left(D_j^*(k+1) \setminus D_j^*(k)\right)$.

Proof: Recall that $M_{j,k}^{(t)} = \min_{i \in D_j^*(k)}(m_i^{(t)})$, it is obvious that $M_{j,k+1}^{(t)} \leq M_{j,k}^{(t)}$. Then we have

$$\begin{aligned}
&C_j(\tilde{D}_j^*(k)) \\
\leq& C_j^*(k-1) + e_w \sum_{t \in T} w_j^{(t)} m_{k+1}^{(t)} \\
&\text{(Reading in the } (k+1)\text{th datacenter may emits less carbon.)} \\
=& C_j^*(k-1) - e_r \sum_{t \in T_{j,k+1}} r_j^{(t)} (m_{k+1}^{(t)} - m_{k+1}^{(t)}) \\
&+ e_w \sum_{t \in T} w_j^{(t)} m_{k+1}^{(t)} \\
=& C_j^*(k-1) - e_r \sum_{t \in T_{j,k+1}} r_j^{(t)} (m_{k+1}^{(t)} - M_{j,k+1}^{(t)}) \\
&+ e_w \sum_{t \in T} w_j^{(t)} m_{k+1}^{(t)} \qquad \text{(Definition of } T_k \text{ and } M_{j,k}^{(t)}) \\
\leq& C_j^*(k-1) - e_r \sum_{t \in T_{j,k+1}} r_j^{(t)} (m_{k+1}^{(t)} - M_{j,k}^{(t)}) \\
&+ e_w \sum_{t \in T} w_j^{(t)} m_{k+1}^{(t)} \qquad (M_{j,k+1}^{(t)} \leq M_{j,k}^{(t)}) \\
=& C_j^*(k-1) - \Delta C_j^*(k+1)
\end{aligned}$$

$\therefore C_j(\tilde{D}_j^*(k)) \leq C_j^*(k-1) - \Delta C_j^*(k+1)$. ∎

Lemma A.3: For all $j \in O$ and $k \leq k_j^*$, $C_j^*(k)$ is a convex function, if $D_j^*(k) \subset D_j^*(k+1)$, where $k = 1, \ldots, |\Omega| - 1$.

Proof: Let $\Delta C_j^*(k) = C_j^*(k-1) - C_j^*(k)$. Since $C_j^*(k)$ is a discrete function, then $C_j^*(k)$ is convex iff $\Delta C_j^*(k) \geq \Delta C_j^*(k+1)$.

Assuming that $\Delta C_j^*(k) < \Delta C_j^*(k+1)$, according to Lemma A.2, we have

$$\begin{aligned}
&C_j(\tilde{D}_j^*(k)) \\
\leq& C_j^*(k-1) - \Delta C_j^*(k+1) \qquad \text{(Lemma A.2)} \\
<& C_j^*(k-1) - \Delta C_j^*(k) \quad \text{(Assuming that } \Delta C_j^*(k) < \Delta C_j^*(k+1)) \\
=& C_j^*(k-1) - C_j^*(k-1) + C_j^*(k) \quad \text{(Definition of } \Delta C_j^*(k+1)) \\
=& C_j^*(k)
\end{aligned}$$

Then this means $C_j(\tilde{D}_j^*(k)) < C_j^*(k)$, which contradict with the definition of $C_j^*(k)$. Therefore $\Delta C_j^*(k) \geq \Delta C_j^*(k+1)$.

Since $C_j^*(k)$, and $\Delta C_j^*(k) \geq \Delta C_j^*(k+1)$, then $C_j^*(k)$ is convex. ∎

Theorem A.4: If $\forall j \in O, D_j^*(k) \subset D_j^*(k+1)$, where $k = 1, \ldots, |\Omega| - 1$, then $C(\mathbf{k}) = \sum_{j \in O} C_j^*(k_j)$ is a convex function, where $\mathbf{k} = (k_1, \ldots, k_{|O|})^T$ and $\forall j = 1, \ldots, |O|$, $k_j \leq k_j^*$.

Proof: Since $C(\mathbf{k}) = \sum_{j \in O} C_j^*(k_j)$, $C(\mathbf{k})$ is a convex function, because it is convex in any dimension, according to Lemma A.3. ∎

Correctness of Fill and Swap.

By running Algorithm 1, follows Algorithm Fill-and-Swap, we can guarantee the returned data replication policy $\mathbf{k}$ is the replication with least cost under the joint-access constraint. It is obvious that if $C \geq \sum_{j \in O} k_j^*$, then Algorithm 1 can find the optimal $\mathbf{k}$. We will prove that if $C < \sum_{j \in O} k_j^*$, our algorithm can find optimal $\mathbf{k}$ by following the gradient of $C(\mathbf{k})$.

Lemma A.5: If $C < \sum_{j \in O} k_j^*$, then the best data replication policy $\mathbf{k}^*$ must fill the whole storage space.

Proof: If the best data replication policy does not fill the storage space, and $C < \sum_{j \in O} k_j^*$, then there must be some data object j, who has less replicas than its optimal number of replicas k_j^*. In this case, we can always add more replicas for the object towards its optimal number of replicas to decrease the cost. This contradict with the definition of the best replication. Hence the best replication must fill the storage space if $C < \sum_{j \in O} k_j^*$. ∎

Theorem A.6: Running Algorithm 1 and Algorithm Fill-and-Swap, we can always find the best data replication policy under the joint-access constraint.

Proof: It is obvious that in both Algorithm 1 and Algorithm Fill-and-Swap, $\forall j \in O, k_j \leq k_j^*$ in anytime. This guarantee that $C(\mathbf{k})$ is convex within the range we consider.

If $C < \sum_{j \in O} k_j^*$, then according to Lemma A.5, we only need to explore replications which can fill the storage space. Once we filled the space in Algorithm Fill-and-Swap, we consider the neighbour point which decrease the most cost. This is basically a gradient descendent algorithm, which can be guaranteed to find a local optimal point of $C(\mathbf{k})$. As we mentioned, $C(\mathbf{k})$ is convex within the range we consider, the local optimal point is also a global optimal point.

If $C \geq \sum_{j \in O} k_j^*$, then it is apparent that Algorithm 1 will find the optimal solution. ∎

References

[1] 1998 World Cup Workload. http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.pdf.

[2] Google Cluster Workload Traces. http://code.google.com/p/googleclusterdata.

[3] Inventory of U.S. greenhouse gas emissions and sinks. http://www.epa.gov/climatechange/Downloads/ghgemissions/US-GHG-Inventory-2013-Main-Text.pdf.

[4] TPC. http://www.tpc.org/.

[5] Wind Integration Dataset. http://www.nrel.gov/electricity/transmission/wind_integration_dataset.html.

[6] Amazon's 'dirty cloud' criticised in greenpeace report. http://www.bbc.com/news/technology-26867362, 2014.

[7] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing. Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 5:1–5:5, New York, NY, USA, 2011. ACM.

[8] S. Almeida, J. a. Leitão, and L. Rodrigues. Chainreaction: A causal+ consistent datastore based on chain replication. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 85–98, New York, NY, USA, 2013. ACM.

[9] Amazon. Amazon s3 pricing. aws.amazon.com/s3/#pricing.

[10] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN)*, pages 9–18, June 2004.

[11] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, 2009.

[12] 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. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[13] N. Deng, C. Stewart, and J. Li. Concentrating renewable energy in grid-tied datacenters. In *ISSST*, pages 1–6. IEEE, 2011.

[14] EIA.GOV. Average retail price of electricity by state. http://goo.gl/UGZhP (Last retrived in Apr., 2014).

[15] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav. It's not easy being green. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 211–222. ACM, 2012.

[16] I. n. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. *SIGARCH Comput. Archit. News*, 41(1):51–64, Mar. 2013.

[17] I. n. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: Leveraging green energy in data-processing frameworks. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 57–70. ACM, 2012.

[18] Google. Green datacenters in google. http://www.google.com/green/energy/.

[19] J. Hwang and T. Wood. Adaptive performance-aware distributed memory caching. In *ICAC*, 2013.

[20] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

[21] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. In *IEEE 2nd International Conference on Cloud Computing Technology and Science*, pages 274–287. IEEE, 2010.

[22] R. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the Sixteenth ACM Symposium on Discrete Algorithms*, SODA '05, pages 630–631. Society for Industrial and Applied Mathematics, 2005.

[23] J. G. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3), 2008.

[24] K. Le, R. Bianchini, T. D. Nguyen, O. Bilgir, and M. Martonosi. Capping the brown energy consumption of internet services at low cost. In *Proceedings of the International Conference on Green Computing*, IGCC '10, pages 3–14, Washington, DC, USA, 2010. IEEE Computer Society.

[25] J. Leverich and C. Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 4:1–4:14. ACM, 2014.

[26] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, pages 233–244. ACM, 2011.

[27] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Dont settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Symposium on Operating Systems Principles (SOSP 11)*, 2011.

[28] A. H. Mahmud and S. Ren. Online capacity provisioning for carbon-neutral data center with demand-responsive electricity prices. *ACM SIGMETRICS Performance Evaluation Review*, 41(2):26–37, 2013.

[29] R. Miller. Where amazons data centers are located, 2008. http://www.datacenterknowledge.com.

[30] J. Paiva, P. Ruivo, P. Romano, and L. Rodrigues. Autoplacer: Scalable self-tuning data placement in distributed key-value stores. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 119–131. USENIX, 2013.

[31] PostgreSQL. http://www.postgresql.org/.

[32] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for internet-scale systems. *SIGCOMM Computer Communication Review*, 39(4):123–134, 2009.

[33] C. Ren, D. Wang, B. Urgaonkar, and A. Sivasubramaniam. Carbon-aware energy capacity planning for datacenters. In *Proceedings of the 2012 IEEE 20th International Symposium on MASCOTS*, MASCOTS '12, pages 391–400. IEEE Computer Society, 2012.

[34] N. L. B. Richard B. Alley, Terje Berntsen and et al. Summary for policymakers: Climate change 2014, mitigation of climate change. *IPCC*, 2014.

[35] D. Sciascia and F. Pedone. Geo-replicated storage with scalable deferred update replication. In *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013*, pages 1–12, June 2013.

[36] P. Shankaranarayanan, A. Sivakumar, S. Rao, and M. Tawarmalani. Performance sensitive replication in geo-distributed cloud datastores. 2014.

[37] C. Stewart and K. Shen. Some joules are more precious than others: Managing renewable energy in the datacenter. In *Proceedings of the Workshop on Power Aware Computing and Systems*, 2009.

[38] G. Szabo and B. A. Huberman. Predicting the popularity of online content. *Commun. ACM*, 53(8):80–88, Aug. 2010.

[39] J. Terrace and M. J. Freedman. Object storage on craq: High-throughput chain replication for read-mostly workloads. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, 2009.

[40] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *Sixth Symposium on Operating Systems Design and Implementation (OSDI '04)*, San Francisco, CA, December 2004.

[41] B. Walsh. Your data is dirty: The carbon price of cloud computing. www.time.com, 2014.

[42] C. Wang, B. Urgaonkar, Q. Wang, G. Kesidis, and A. Sivasubramaniam. Data center power cost optimization via workload modulation. In *Proc. of UCC*, 2013.

[43] W. Wang, B. Li, and B. Liang. To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 13–22. USENIX, 2013.

[44] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 292–308. ACM, 2013.

[45] H. Xu, C. Feng, and B. Li. Temperature aware workload management in geo-distributed datacenters. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 303–314, San Jose, CA, 2013. USENIX.

[46] Z. Xu, Y. Tu, and X. Wang. Dynamic energy estimation of query plans in database systems. In *IEEE 33rd International Conference on Distributed Computing Systems, ICDCS*, pages 83–92, 2013.

[47] Yahoo! Yahoo! labs webscope, 2010. webscope.sandbox.yahoo.com.

[48] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. In *Proceedings of the 1st ACM EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 333–344, 2006.

[49] Y. Zhang, Y. Wang, and X. Wang. GreenWare: Greening Cloud-scale Data Centers to Maximize the Use of Renewable Energy. In *Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware*, Middleware'11, pages 143–164, 2011.