Systems Support for Carbon-Aware Cloud Applications

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Nan Deng, B.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2015

Abstract

Datacenters, which are large server farms, host *cloud applications*, providing services ranging from search engines to social networks and video streaming services. Such applications may belong to the same owner of the datacenter or from third party developers. Due to the growth of cloud applications, datacenters account for a larger fraction of worldwide carbon emissions each year. To reduce the carbon emissions, many datacenter owners are slowly but gradually adopting clean, renewable energy, like solar or wind energy.

To encourage datacenter owners to invest into renewable energy, the usage of renewable energy should lead to profit. However, in most cases, renewable energy supply is intermittent and may be limited. Such fact makes renewable energy more expensive than traditional dirty energy. On the other hand, not all customers have the need of using renewable energy for their applications. Our approach is to devise accountable and effective mechanisms to deliver renewable energy only to users that will pay for renewable-powered services. According to our research, datacenter owners could make profit if they could concentrate the renewable energy supply to *carbon-aware applications*, who prefer cloud resources powered by renewable energy. We develope two carbon-aware applications as use cases. We conclude that if an application take carbon emissions as a constraint, it will end up with using more resources from renewable powered datacenters. Such observation helps datacenter owners to wisely distribute renewable energy within their systems.

Our first attempt of concentrating renewable energy focuses on architectural level. Our approach requires datacenters have on-site renewable energy generator using *grid ties* to integrate renewable energy into their power supply system. To measure the concentration of renewable energy, we introduce a new metric, the *renewable-powered instance*. Using this metric, we found that grid-tie placement has first-order effects on renewable-energy concentration.

On-site renewable energy requires an initial investment to install renewable generator. Although this cost could be gradually amortized over time, some people prefer renewable energy credit, which could be bought from utility companies by paying premium for the renewable energy transmitted through the grid and produced in other locations. To let datacenters, with or without on-site renewable energy generator, attract more carbon-aware customers, we designed a system for *Adaptive Green Hosting*. It identifies carbon-aware customers by signaling customers' applications when renewable energy is available and observing their behaviors. Since carbon-aware applications would tend to use more resources in a datacenter with low emission rates, datacenter owners could make profit by attributing more renewable energy to carbon-aware applications, so that could encourage them to use more resources. Our experiments show that adaptive green hosting can increase profit by 152% for one of todays larger green hosts.

Although it is possible for cloud applications to maintain a low carbon footprint while make profit, most existing applications are not carbon-aware. The carbon footprint for most existing workloads is large. Without forcing them to switch to renewable energy, we believe responsible end users could take a step forward first. We propose a method to help end users to discover implementation-level details about a cloud application by extracting its internal software delays. Such details are unlikely to be exposed to third-party users. Instead, our approach probes target application from outside, and extract normalized software delay distributions using only response times. Such software delay distributions are not only useful to reveal normalized energy footprint of an application, but could also be used to diagnose root causes of tail response times for live applications.

To my parents, my wife and my mentor.

Acknowledgments

I enjoy my journey to study in the United States. I could never make this far without everyone's help. It is never an easy decision for parents to encourage their child to travel across the world and study for years in a foreign country. I appreciate my parents' vision and their support. It was their early education that keep encouraging me to be curious to the outside world. I am also especially thankful for my wife Ziqi. Working in the same department, she is my first reader, best reviewer. I am thankful for her work on our wedding ceremony while I was writing my thesis.

I am grateful to have worked with my advisor Dr. Christopher Stewart, who provided guidance for research and personal advice. His insights encouraged me to dig deeper into the problems and his kindness could always relieve my stress when I was desperate for a solution. I also thank Dr. Xiaorui Wang and Dr. Gagan Agrawal for serving on my committee.

It is my honor to have made so many friends in our department. Thanks for their help and support: Zichen Xu, Jaimie Kelley, Aniket Chakrabarti, Nathaniel Morris, Zhezhe Chen, Mai Zheng, Dachuan Huang, Yang Zhang, Yuxuan Wang and Man Cao.

Special thanks to my dog QiuQiu, a small Yorkshire Terrier with a big heart. He trained me to be patient and smart. Thanks for his everyday whining that reminds me to do my daily exercise even in the most busy days.

Vita

2009	B.S. Telecommunication Engineering, Beijing University of Post and Telecommunication
2010-Present	Graduate Teaching Associate, Department of Computer Science and Engineering, The Ohio State University
2010-Present	Graduate Research Associate, Department of Computer Science and Engineering, The Ohio State University

Publications

Research Publications

Zichen Xu, Nan Deng, Christopher Stewart, Xiaorui Wang. Blending On-Demand and Spot Instances to Lower Costs for In-Memory Storage. In *Proceedings of 2016 IEEE International Conference on Computer Communications (INFOCOM'16)*, 2016. *Under submission.*

Nan Deng, Zichen Xu, Christopher Stewart, Xiaorui Wang. Tell-Tale Tails: Decomposing Response Times for Live Internet Services. In *Proceedings of the 6th International Green and Sustainable Computing Conference (IGSC'15)*, 2015.

Zichen Xu, Nan Deng, Christopher Stewart, Xiaorui Wang. CADRE: Carbon-Aware Data Replication for Geo-Diverse Services. In *Proceedings of the 12th IEEE International Conference of Autonomic Computing (ICAC'15)*, 2015.

Nan Deng, Zichen Xu, Christopher Stewart, Xiaorui Wang. From the Outside Looking In: Probing Web APIs to Build Detailed Workload Profiles. In *Proceedings of the 9th International Workshop on Feedback Computing*, 2014.

Nan Deng, Christopher Stewart, Jaimie Kelley, Daniel Gmach and Martin Arlitt. Adaptive Green Hosting. In *Proceedings of the 9th ACM International Conference on Autonomic Computing (ICAC'12)*, 2012.

Nan Deng, Christopher Stewart, Daniel Gmach, and Martin Arlitt. Policy and Mechanism for Carbon-Aware Cloud Applications. In *Proceedings of 2012 IEEE/IFIP Network Operations and Management Symposium (NOMS'12)*, 2012.

Nan Deng, Christopher Stewart. Concentrating Renewable Energy in Grid-Tied Datacenters. In *Proceedings of 2011 IEEE International Symposium on Sustainable Systems and Technology (ISSST'11)*, 2011.

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Over the next 5 years, the number of Internet users will grow by 60% and cloud applications will host 3X more data [102]. As demand increases, carbon emissions from those applications will grow. Already, the global IT sector emits more carbon dioxide than most countries [84]. By 2020, the datacenters that host cloud applications will emit 340 metric megatons of CO_2 annually, twice as much as The Netherlands [61]. The growing emissions rate of the IT sector is happening at time when carbon emissions worldwide must decrease by 72% to prevent climate change [43]. The enormity of this challenge has prompted many IT companies to voluntarily reduce their emissions. A growing cadre of green datacenters now use rooftop solar panels and on-site wind turbines [5, 41], to reduce their carbon footprint.

On the other hand, a small group of users are starting to realize the importance of low-carbon computing, gradually forming a demand of a niche market. For example, over 700,000 Facebook members have signed a petition pressuring the site's managers to use only renewable energy [99] in 2010. After 20 months, Facebook agreed to invest in renewable energy [100]. Similarly, Greenpeace.org used an industry-wide study of carbon emissions to put pressure on a wide range of IT companies [27]. Increasingly, hosting services, i.e., companies that lease datacenter resources to cloud applications, exploit user demand for low-carbon computing by conspicuously powering their infrastructure with renewable energy. Examples of such green hosts include AISO [1], HostGator [56], Green Geeks, and GreenQloud [53]. Green hosts attract carbon-aware users. AISO, the eldest of these green hosts, enjoyed 10% CAGR between 2002 and 2008 [101].

Using clean, renewable energy to power the datacenters can reduce carbon emissions. However, renewable energy may increase energy costs compared to traditional, dirty energy. Increased costs have prevented wide adoption of renewable energy within datacenters. Our work promotes low-carbon computing by helping datacenter owners and Internet application providers keep costs low while delivering clean energy to end users. With our holistic approach, low-carbon computing will be not only be cost effective but profitable as well.

1.1 Thesis Statement

Our thesis is that *accounting for and managing the carbon footprint of carbon-aware users can increase profit for datacenters and cloud applications while reducing the carbon footprint of Internet services.* Our research builds upon recent studies that have shown that carbon-aware users are an under served niche market [85].

Figure 1.1 depicts the parties involved in carbon-aware management. Our approach builds upon the traditional cloud computing market. Cloud applications lease resources from datacenters and provide services to end users. Our research considers the carbon footprint of each party. To quantify the carbon footprint, we use the term *carbon offset* to represent a unit of clean energy that can replace a unit of dirty energy, both measured in Joules. Carbon offsets can be produced by on-site solar panels,

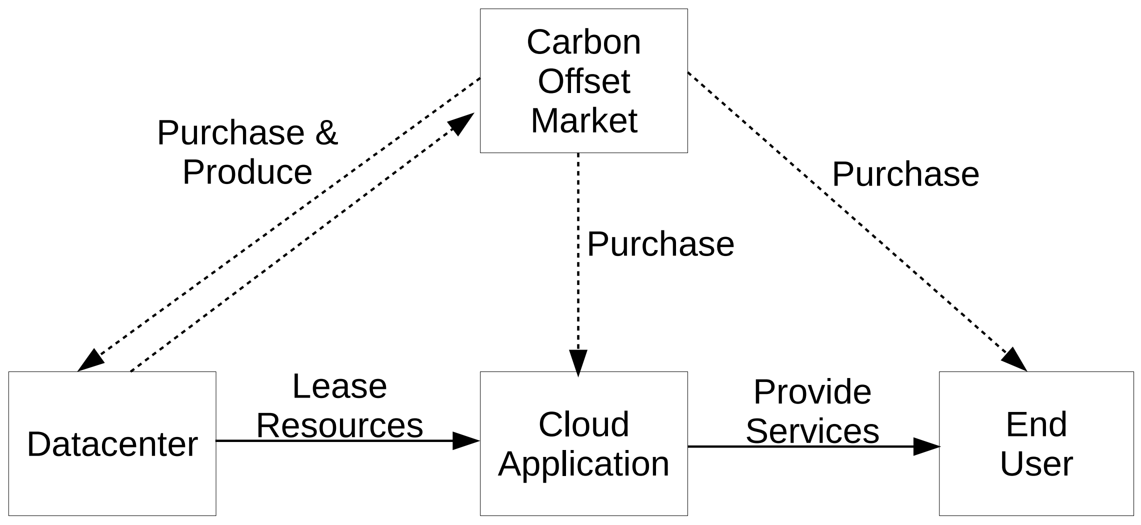


Figure 1.1: We propose carbon-aware management for (1) datacenters and cloud applications that puchase carbon offsets, (2) datacenters that produce renewable energy on-site and (3) end users with limited visibility to system backends.

power received from local wind farms, or renewable energy credits (RECs) purchased via energy markets. Datacenters could purchase carbon offsets to reduce the carbon emissions caused by their harware. Alternatively, datacenters could use on-site solar panels or wind turbines to produce carbon offsets. In our model, cloud applications and end users can only consume carbon offsets. Specifically, carbon-aware users, including application owners and end users, may wish to maintain a certain *offset ratio* — i.e., the ratio of carbon offsets to dirty energy. Our work builds on the following axiomatic assumptions.

- *Cloud computing decouples the resource providers and cloud application owners.* Computing and storage resources are leased to cloud applications for a fixed period. The startup cost of building and installing renewable energy generators, like solar panel or wind turbine, is amortized and embedded in the price when leasing cloud resources. This lowers the bar for cloud application owners to step into the market.

- *Carbon-aware customers would take carbon footprint into account when they choose where to host their applications.* Some carbon-aware customers may even like to pay a premium to use low-emissions resources [25]. This provides an opportunity for cloud providers to attract carbon-aware customers by providing profitable low footprint resources.

- *An increasing number of end users care about clean computing.* This would push companies to move away from heavy carbon footprint resources. As we mentioned above, Facebook has already announced to invest into renewable energy partly because of a petition by its users [100].

1.2 Contributions

Figure 1.2 depicts our contributions at each layer in the cloud computing stack. Power stations generate electricity power using either dirty or clean energy sources. The grid delivers a mix of renewable and dirty energy to the datacenters' power delivery system. Some datacenters may have their own on-site renewable energy generator, which will deliver pure renewable energy to the power delivery system. The power delivery system provides energy supply to support computing and storage resources inside a datacenter. One of our contributions is an analysis on where to integrate renewable energy into datacenters to maximize the concentration of renewable energy. Datacenters lease their computing and storage resources to their customers upon request. Cloud applications deployed by datacenters' customers use these cloud

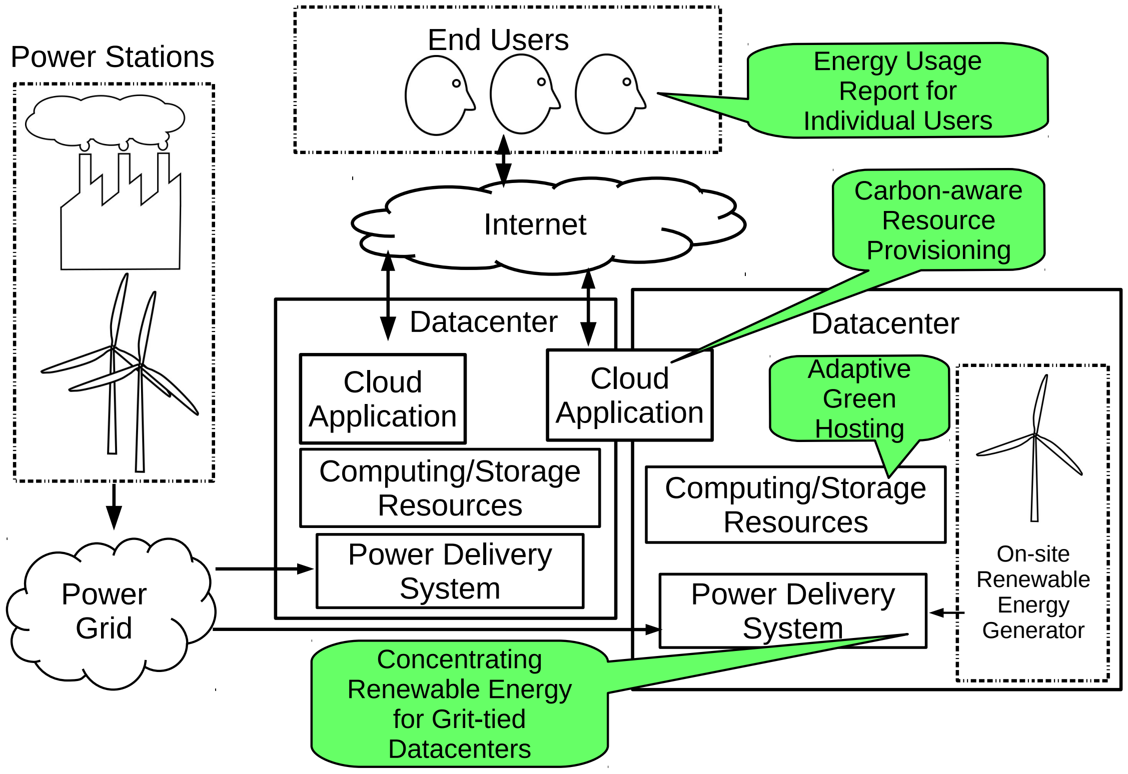

Power Stations
End Users
Energy Usage Report for Individual Users
Internet
Carbon-aware Resource Provisioning
Datacenter
Cloud Application
Computing/Storage Resources
Power Delivery System
Datacenter
Cloud Application
Adaptive Green Hosting
Computing/Storage Resources
Power Delivery System
On-site Renewable Energy Generator
Power Grid
Concentrating Renewable Energy for Grit-tied Datacenters

Figure 1.2: Overview

resources from datacenters and provide services through Internet for their end users. Some cloud applications may use resources from different datacenters. Several factors — like number of cores, datacenter location, etc. — affect cloud applications' decision of where to lease cloud resources.

1.2.1 Carbon-aware Resource Provisioning

To understand carbon-aware customers' behavior, we first studied several carbon-aware applications running in geo-diverse datacenters. In general, carbon-aware applications deploy their resources and/or direct their workloads by solving optimization problem, which takes carbon footprint into account.

We build two carbon-aware applications: One application reduces its carbon footprint by scaling its computing resources within several datacenters; Another one reduces its carbon footprint by allocating its storage resources from differerent datacenters. Both of them could effectively reduce their carbon emissions without compromising the performance.

1.2.2 Concentrating Renewable Energy in Datacenters

We studied an architectural approach for datacenters with on site renewable energy generators. We argue that renewable energy should be concentrated as much as possible on the servers used by carbon-aware customers. To measure the concentration, we introduce a new metric, the *renewable-powered instance*, that measures the concentration of renewable energy in datacenters.

We conducted a simulation-based study of renewable-energy datacenters, focusing on the grid tie—the device most commonly used to integrate renewable energy. We found that grid-tie placement has first-order effects on renewable-energy concentration.

1.2.3 Adaptive Green Hosting

For datacenters without on-site renewable energy, renewable energy credit could be another option. Datacenter owners could buy renewable energy credit from their utility providers to offset emission caused by their electricity energy usage. Although additional cost is introduced to buy such credit, datacenter owners could use it to make profit by attracting more datacenter resource usage from carbon-aware customers.

We designed *Adaptive Green Hosting* service, which identifies carbon-aware applications by signaling customers' applications when renewable energy is available and observing their behaviors. Datacenter owners could make profit by finding and offering the best emission rate for each application to maximize their resource usage.

1.2.4 Energy Usage Report for End Users

We believe that the demand of Internet application end users is the ultimate motivation for Internet applications and datacenters to switch to low carbon energy. Letting users know an application's carbon footprint is the first step. However, due to security and commercial considerations, application owners are very unlikely to disclose their energy footprints in details.

To help users understand a third-party application, we created an approach to decompose an Internet application's response times into per-component delays and then extract normalized energy footprint of each component. Using our method, users are able to analyze an application's per-request energy footprint. We further found that our approach is even useful to analyze an application's tail performance and its internal software stack.

Chapter 2: Carbon-aware Applications

Datacenters provide resources for customers to run their cloud applications. Cloud applications provision their resources based on their workload and pay to datacenter owners according to how much resources they have used. Carbon-aware applications consider emission rates of a datacenter as an additional factor when they choose where to provision their resources. In this chapter, we study two carbon-aware applications. One is a general purpose cloud application, which deploys its resources within cloud instances. The second one is a storage application, which replicates data to sites that combine together to yield low carbon footprint.

2.1 Policy and Mechanism for Carbon-Aware Cloud Applications

Carbon-aware cloud applications run inside of a renewable-energy datacenter, provision resources on demand, and seek to minimize their use of carbon-heavy, grid energy. In this section, we first introduce a provisioning policy that address the uncertainty of renewable energy. Then, we discuss mechanisms for datacenters to determine the contribution of grid energy to specific application workloads.

2.1.1 Policy: Carbon-Aware Provisioning

Before introducing the carbon-aware provisioning problem, we first define a few terms for readers without a background in cloud computing. A *cloud resource* is a collection of computing hardware within a datacenter, e.g., CPU, disk, and memory. A cloud resource running for a fixed period of time is a *cloud instance* . An *application manager* provisions cloud instances that collectively compose a *cloud application* .

In carbon-aware provisioning, the application manager sets a hard limit on the total carbon-heavy, grid energy of provisioned cloud instances. Carbon-aware policy decides which cloud instances to provision such that 1) the grid energy limit is not exceeded and 2) performance goals are met. In this paper, we focus on throughput (i.e., requests per second or job processing rate) as performance metrics. Carbon-aware policy needs estimates of grid-energy needs, performance goals, and grid-energy caps. These can be adjusted at each time interval where cloud instances can be provisioned.

Suppose there are n cloud instances available for provisioning. At time period t, a provisioning strategy for an application is denoted as a vector $\mathbf{x^{(t)}} = (x_1^{(t)}, x_2^{(t)} \cdots x_n^{(t)})^T$, in which $x_i^{(t)} = 1$ if the ith instance is provisioned, otherwise $x_i^{(t)} = 0$. Each cloud instance can provide up to its *maximum throughput* for a target application, represented by the vector $\mathbf{v} = (v_1, v_2 \cdots v_n)^T$. v_i is a real number that denotes the maximum throughput provided by the server which will run ith instance. The *maximum throughput* of a cloud application is the summation of the maximum throughput of its provisioned instances, i.e. $\mathbf{v}^T\mathbf{x^{(t)}}$. Here, the symbol T is the transpose function in vector multiplication. We also note that some applications, e.g., Web servers, may have fluctuating throughput needs over time. We use $V^{(t)}$ to represent the target throughput at time t.

The hard limit on carbon-heavy, grid energy is $D^{(t)}$ as set by the application manager. The total grid energy used by an instance at time t is represented as a vector $\mathbf{d^{(t)}} = (d_1^{(t)}, d_2^{(t)} \cdots d_n^{(t)})^T$. Note, $d_i^{(t)}$ is a real number between zero and max energy needs of the ith instance. An application's total grid energy consumption at time t is $\mathbf{d}^T\mathbf{x^{(t)}} < D^{(t)}$.

Given $\mathbf{v}, \mathbf{d^{(t)}}, D^{(t)}, V^{(t)}$, our goal is to find $\mathbf{x^{(t)}}$. Specifically, we represent carbon-aware provisioning as an integer programming problem:

$$\text{Maximize } \mathbf{v}^T\mathbf{x^{(t)}} \tag{2.1}$$

$$\text{Subject to } \mathbf{d^{(t)}}^T\mathbf{x^{(t)}} \leq D^{(t)} \tag{2.2}$$

$$\text{and } \mathbf{v}^T\mathbf{x^{(t)}} \leq V^{(t)} \tag{2.3}$$

This problem is a variant of the well-known NP-complete knapsack problem [58]. A deterministic algorithm that finds optimal provisioning strategies quickly (i.e., in polynomial time) is unlikely. However, algorithms that quickly find approximate solutions exist [58]. Figure 2.1 shows a recursive dynamic programming solution [75]. Here, the function $f(i, j)$ means the maximum throughput (less than $V^{(t)}$) of the application if we only consider the first i instances. The initial value of j is the manager's limit on carbon-heavy energy.

The boundary condition is $f(0, 0) = 0$. The dynamic programming algorithm uses 2-level loop, iterating from $i = 0$ to $i = n$; and in each i loop, iterating j from 0 to $D^{(t)}$. The algorithmic runtime for this solution is $O(nD^{(t)})$.

$$f(i,j) = \begin{cases} f(i-1,j) & \text{if } d_i^{(t)} > j \\ & \text{or } d_i^{(t)} \le j \\ & \quad \text{and } f(i-1,j) > v_i + f(i-1,j-d_i^{(t)}) \\ & \text{or } d_i^{(t)} \le j \\ & \quad \text{and } f(i-1,j) \le v_i + f(i-1,j-d_i^{(t)}) \\ & \quad \text{and } f(i-1,j-d_i^{(t)}) + v_i > V^{(t)} \\ & \quad \text{and } f(i-1,j) > f(i,j-1) \\ f(i,j-1) & \text{if } d_i^{(t)} \le j \\ & \text{and } f(i-1,j) \le v_i + f(i-1,j-d_i^{(t)}) \\ & \text{and } f(i-1,j-d_i^{(t)}) + v_i > V^{(t)} \\ & \text{and } f(i-1,j) \le f(i,j-1) \\ f(i-1,j-d_i^{(t)}) + v_i & \text{otherwise} \end{cases}$$

Figure 2.1: Dynamic Programming Solution

In renewable-energy datacenters, the grid energy used by each cloud resource changes unpredictably with the weather. When the sun is blocked by clouds, solar panels produce less electricity, and the grid delivers more energy to cloud resources. That is, each resource's dependence on carbon-heavy energy ($\mathbf{d^{(t)}}$) could change at any moment. Neither recent history, nearby weather, nor cyclic patterns can predict $\mathbf{d^{(t)}}$ perfectly [87]. Inaccurate predictions will affect carbon-aware policy.

Extending our earlier problem definition, let $\hat{d_i^{(t)}}$ denote the actual grid-energy usage by ith instance at time t. Let $\Delta d_i^{(t)}$ be a random variable denoting the difference between actual grid-energy usage at time t ($\hat{d_i^{(t)}}$) and the predicted grid-energy usage at time t ($d_i^{(t)}$). We use vectors $\mathbf{\Delta d^{(t)}}$ and $\mathbf{\hat{d^{(t)}}}$ to represent $(\Delta d_1^{(t)}, \Delta d_2^{(t)} \cdots \Delta d_n^{(t)})^T$

and $(\hat{d_1^{(t)}}, \hat{d_2^{(t)}} \cdots \hat{d_n^{(t)}})^T$, respectively. Our new extended problem definition is:

$$\text{Maximize } \mathbf{v}^T\mathbf{x^{(t)}} \tag{2.4}$$

$$\text{Subject to } \hat{\mathbf{d}^{(\mathbf{t})}}^T\mathbf{x^{(t)}} - \Delta\mathbf{d^{(t)}}^T\mathbf{x^{(t)}} \leq D^{(t)} \tag{2.5}$$

$$\text{and } \mathbf{v}^T\mathbf{x^{(t)}} \leq \hat{V}^{(t)} + \Delta V^{(t)} \tag{2.6}$$

From Equation (2.5), we can see that once we make the decision of our strategy, the real upper bound of the dirty energy used is $D^{(t)} + \Delta\mathbf{d^{(t)}}^T\mathbf{x^{(t)}}$. Hence $\Delta\mathbf{d^{(t)}}^T\mathbf{x^{(t)}}$ is the error we may introduce. Using this model of uncertainty as a base, we plan to answer the following research questions:

1. Which probability distribution characterizes $\Delta d_i^{(t)}$*?* The answer depends on the prediction method [87] and the on-site renewable-energy source (e.g., solar versus wind). However, both solar and wind energy are subject to sudden unexpected outages that lead to heavy-tail production patterns [94]. Heavy-tail production often leads to heavy-tail prediction error also.

2. Do different instances have different $\Delta d_i^{(t)}$*?* If on-site renewable energy is evenly distributed to all resources, we would expect the answer to be no. However, our recent work [33] shows that grid-tie placement can concentrate renewable energy to certain datacenter resources. Our results shown later in this section confirm this result using a different mechanism. If the answer is yes, $\Delta d_i^{(t)}$ may be statistically dependent between instances.

Note that our model naturally extends to the practical case where instances are heterogeneous. However, in the simple homogeneous case—if renewable energy is evenly distributed— then we can assume that $\Delta\mathbf{d^{(t)}}$ are independent and identically distributed random variables. If there are large enough number of instances running the service, it is safe to say that $\Delta \mathbf{d^{(t)}}^T \mathbf{x^{(t)}}$ is a normally distributed random variable according to central limit theory. This condition may be important for datacenters that apply renewable energy incrementally by first targeting a few homogeneous racks.

2.1.2 Mechanism: Carbon Accounting

The carbon-aware policy discussed in Section 2.1.1 supports grid-tied datacenters where the electric grid contributes fractionally to energy needs. These contributions were $d_i^{(t)}$, a parameter that could range from 0 to the energy consumption of resource i. Carbon accounting is the mechanism that attributes carbon-heavy electricity to cloud resources. As we began to implement carbon-aware applications, we found that grid ties complicate carbon accounting.

The DC electricity produced by solar panels is incompatible with the AC electricity pulled from the electric grid. Grid ties solve this problem by inverting DC electricity from on-site sources and producing AC electricity that matches the grid's voltage and frequency. Electricity from these two sources become indistinguishable to datacenter devices which draw power as they normally would. Grid ties are necessary for net metering, a widely used concept in which owners of solar panels and wind turbines can be reimbursed for contributing electricity to the grid— a potential revenue stream for datacenter owners. Also, the cost of grid ties does not increase at scale, unlike, batteries for energy storage. Finally and critically, grid ties can be deployed incrementally. Allowing datacenters to slowly invest in renewable energy. However, grid ties make it difficult to infer the exact contribution of carbon-heavy sources. Accurate and technical carbon accounting must underlie any practical carbon-aware system.

Our proposed solution measures the net energy pulled from the grid before and after grid ties are installed. Since grid ties do not affect the energy needs of datacenter devices, the amount of electricity flowing through power-delivery devices upstream relative to grid ties must decrease (according Kirchhoff's current law [80]). The proportional sharing principle [20] attributes energy from multiple sources to individual sinks (i.e., cloud resources) according to each source's relative contribution to total energy needs. In our solution, the grid contributes the net grid energy pulled *after* a grid tie is installed. The total energy needs are the net grid energy pulled *before* a grid tie is installed.

As we will discuss in the later chapter, under a net-energy model, grid-tie placement affects the concentration of grid energy used to power cloud resources [33]. A fixed injection of electricity causes greater relative reduction in grid energy when there are few devices downstream (i.e., a small denominator). A grid tie could inject more electricity on the circuit than the downstream resources need. In this case, the electricity flowing through power delivery devices further upstream decreases recursively.

2.1.3 Experiment Results

Combining policy and mechanism, we have set up an experimental renewable-energy cluster in our lab (shown in Figure 2.2). The top component in Figure 2.2 is a power supply unit with capacity that easily exceeds the max power of our cluster. This power supply 1) serves as our experimental grid and 2) isolates electricity produced by our grid tie (ensuring proper, downstream-only usage of the real grid). The bottom component is a programmable power supply that connects to a grid tie; it can be replaced by a real solar panel (i.e., for demo). Figure 2.2 also shows how these

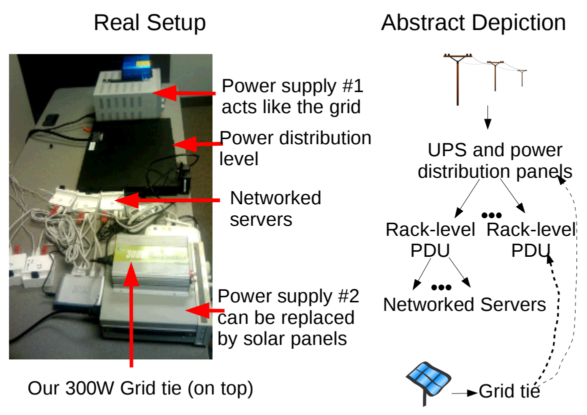

Real Setup
Abstract Depiction
Power supply #1 acts like the grid
Power distribution level
Networked servers
Power supply #2 can be replaced by solar panels
Our 300W Grid tie (on top)
UPS and power distribution panels
Rack-level PDU
Rack-level PDU
Networked Servers
Grid tie

Figure 2.2: Our renewable-energy cluster; the real setup and abstract goal. Dotted lines reflect grid-tie placements. Note to reviewer, we would like to host a demo at the conference.

components relate to traditional power delivery within the datacenter. It allows us to produce repeatable results. Our cluster has 5 low-power Linux servers, each with Apache and MapReduce capabilities, a 1GbE switch, and a 1GbE router. Collectively, our cluster consumes about 42W, excluding the power supply. We used clamp-on amp meters and line splitters to measure the power usage at each link.

Mechanism Experiments: A cloud resource in our renewable-energy cluster is 1 server (1.2Ghz processor, 128 MB, 1 Gb Ethernet). Power delivery devices in our renewable-energy cluster are power strips and 1 rack-style PDU. Our grid tie needs and produces 120V AC, making it compatible with our servers. This allows us to measure the effect of grid-tie placement on net grid energy. Here, we present an experiment that compares two grid-tie placements. The first places the grid tie just below our grid-like power supply, a top-level placement that evenly dilutes the concentration of energy from on-site sources [33]. The second places the grid tie on a circuit where cloud resources 2–4 are downstream, but resources 0–1 are powered in

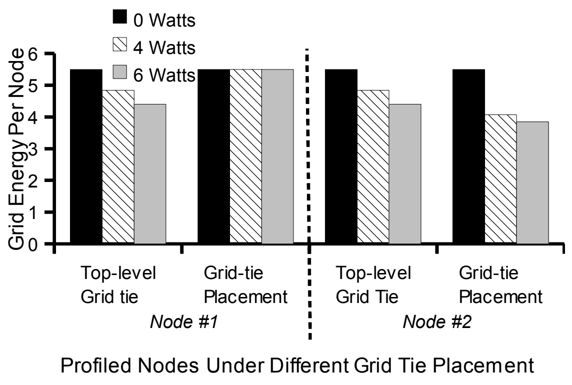


Figure 2.3: Net grid energy used by nodes 1 and 2 in our cluster. The legend shows the amount of power injected by the grid-tied power source.

parallel on a different circuit. This changes the net contribution of renewable energy between the two groups. Figure 2.2 depicts grid-tie placement.

Figure 2.3 shows the grid energy used by nodes 1 and 2 in our experiment. We injected different amounts of “renewable” energy using our programmable power supply. The grid-tie placement changed the distribution of renewable energy under our carbon accounting approach. The low-level placement sent more renewable energy to nodes 2–4 by taking from nodes 0–1, increasing the sustainability of node 2 by 14%.

Policy Experiment: We set up the Apache Web Server [3] on our cluster. The workload matched a scaled version of the World Cup 1998 logs [10]. We provisioned instances by restarting Apache on a cluster node and telling our load balancer to direct requests to it. We unprovisioned instances by stopping Apache. The maximum throughput of each cloud resource (v_i) was measured in requests per second (rps) via offline profiling. Our homogeneous cluster nodes supported 15 rps before the average response time exceeded our limit of 100ms. We also considered a *heterogeneous* cluster where hypothetical profiling provided the same maximum throughput (75 rps) but

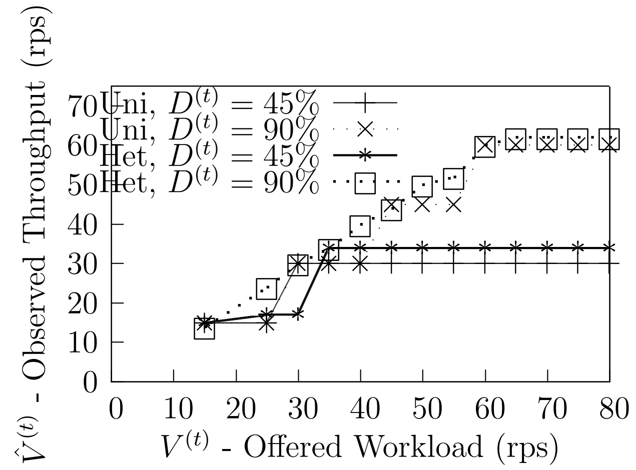


Figure 2.4: Observed throughput versus offered workload for Apache under a carbon-heavy energy cap. Uni is the homogeneous cluster; Het stands for heterogeneous.

with varied rps per node (from 12–20). Building on Figure 2.3, we configured our cluster for a grid-tie placement that gave more renewable energy to nodes 2–4 under our operational carbon accounting. We set the contribution of renewable energy to 10% ($D^{(t)}$ is high) and 55% ($D^{(t)}$ is low) of the cluster's maximum energy.

Figure 2.4 has important implications for application managers concerned about sustainability. First, the heterogeneous cluster achieves higher throughput than the homogeneous. This is because the heterogeneous cluster supports diverse combinations of instances, and our carbon-aware policy naturally controls this parameter. Second, strong (low) carbon caps degraded performance proportionally relative to the cap, whereas weak (high) carbon caps had disproportionate effects. Under a low carbon cap (45% of total power), the observed throughput under the heterogeneous cluster saturated at 43% of the maximum. However, under a high carbon cap (90% of total power), the observed throughput dropped to only 82% of the maximum. Here again, low carbon caps allowed our integer-programming approach to choose from more instance combinations. These preliminary results indicate that an application looking to reduce its carbon footprint should prefer datacenters that supply diverse instances.

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

In this section, we consider another carbon-aware application named *CADRE, a carbon-aware data replication approach for geo-diverse services.* The service tries to minimize carbon emissions induced by subsequent read and write operations of content, the former (read) involving a single replica at the datacenter with lowest emission rates and the latter (write) involving all replicas. The application considers the read-write ratio of a data object and find the best number of replicas under the predicted carbon emission rates traces for all datacenters.

2.2.1 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. (2.7), (2.8), and (2.9).

$$C_{j,r}(D_j) \triangleq \sum_{t=T_1}^{T_2} \sum_{q \in Q} (e_r + z_{i,q}^{(t)}) r_{j,q}^{(t)} \min_{i \in D_j}(m_i^{(t)}) \tag{2.7}$$

$$C_{j,w}(D_j) \triangleq e_w \sum_{t=T_1}^{T_2} w_j^{(t)} \sum_{i \in D_j} m_i^{(t)} \tag{2.8}$$

$$C_j(D_j) \triangleq C_{j,r}(D_j) + C_{j,w}(D_j) \tag{2.9}$$

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

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 2.1: Symbols and notations. **Bold** indicates vectors.

read. All notations are defined in Table 2.1. The models use average emission and access rates over discrete intervals. 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 [105]. 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 [63].

2.2.2 Footprint-Replication Curves

Replication helps dispatchers reduce footprints on read accesses, but it increases the footprints of write accesses. Using Eq. (2.9), we explore an object's total carbon footprints under all possible replication factors. The graph, where each point on the

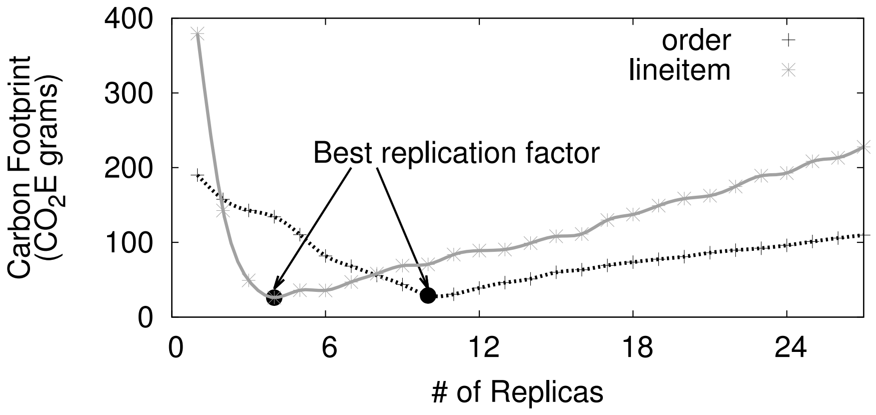


Figure 2.5: The convex footprint-Replication curves.

y-axis is the smallest carbon footprint achievable under the corresponding replication factor on the x-axis, is *a footprint-replication curve.* More precisely, $\mathbf{R}_j$: $R_{j,k} = C_j^*(k)$.

Figure 2.5 gives an example of two footprint-replication curves (i.e., lineitem and order data tables in TPC-H workloads [7]). 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 2.5 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.

Formally, the curve contains points which are solutions to the following optimization problem under different ks:

$$D_j^*(k) \triangleq \underset{D \subseteq \Omega_j, |D|=k}{\arg\min} \; C_j(D) \tag{2.10}$$

$$D_j^*(k-1) \subset D_j^*(k) \tag{2.11}$$

$$k_j^* = \underset{k=1,\ldots,|\Omega|}{\arg\min} D_j^*(k) \tag{2.12}$$

$$C_j^*(k) \triangleq C_j(D_j^*(k)) \tag{2.13}$$

Eq. (2.10), (2.11), (2.12), and (2.13) present optimization models for the footprint-replication curve.

Eq. (2.11) 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 [62]. 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. We call it *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.

Lemma 2.2.1. *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.

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

□

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.

2.2.3 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 ratio 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 2.2.2. *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 $\tilde{D}_j^*(k) = D_j^*(k-1) \cup \big(D_j^*(k+1) \setminus D_j^*(k)\big)$.*

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

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

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

Lemma 2.2.3. *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 2.2.2, we have

$$\begin{aligned}
&C_j(\tilde{D}_j^*(k)) \\
\le& C_j^*(k-1) - \Delta C_j^*(k+1) && \text{(Lemma 2.2.2)} \\
<& C_j^*(k-1) - \Delta C_j^*(k) && \text{(Assuming that } \Delta C_j^*(k) < \Delta C_j^*(k+1)\text{)} \\
=& C_j^*(k-1) - C_j^*(k-1) + C_j^*(k) && \text{(Definition of } \Delta C_j^*(k+1)\text{)} \\
=& C_j^*(k)
\end{aligned}$$

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

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

Theorem 2.2.4. *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* $\forall j = 1, \ldots, |O|$, $k_j \le k_j^*$.

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 2.2.3. □

2.2.4 Finding the Optimal Solution

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. (2.14), managers can set the *provisioning factor* f to force partial replication because full

replication is often too costly.

$$\sum_{|O|} k_j \leq |O||\Omega| f \tag{2.14}$$

2. Availability: Replication to geo-diverse sites ensure that objects are durable and available during earthquakes, fires and other regional outages. Eq. (2.15) ensures that every object is replicated to at least K sites but no more than $|\Omega_j|$.

$$K \leq k_j \leq |\Omega_j|, \forall j \leq |O| \tag{2.15}$$

3. Load Balancing: Consistent hashing uses virtual sites to handle heterogeneity. CADRE overrides these settings to consider time varying emission rates. Managers can devalue emission rates by setting per-site weights, shown in Eq. (2.16). The weight vector can be changed on the fly.

$$m_i^{(t)} \triangleq m_i^{*(t)} w_i. \tag{2.16}$$

To find the optimal solution, we will 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.

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 greedy online algorithm.

Algorithm 1 Greedy Online Algorithm

```
1: $C_{remain} := |O||\Omega|f$
2: for $j := 1$ TO $|O|$ do
3:     if $k_j^* \leq K$ then
4:         //Availability constraint
5:         $C_{remain} := C_{remain} - K$
6:         $k_j := K$;
7:     else if $C_{remain} - k_j^* > (|O| - j)K - K$ then
8:         $C_{remain} := C_{remain} - k_j^*$ //Replicating to optimum
9:         $k_j := k_j^*$
10:    else
11:        $C_{remain} := C_{remain} - K$ //Replicating to minimum
12:        $k_j := K$
13:    end if
14: end for
```

If there always exist spare capacities for data replication, It is obvious that the greedy algorithm is already optimal. Formally, if $C \geq \sum_{j \in O} k_j^*$, then Algorithm 1 can find the optimal $\mathbf{k}$. 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*. 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.

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. As we mentioned 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 2.2.5. *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 2.2.6. *Running Algorithm 1 and 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 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 2.2.5, 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.

$\square$

2.3 Discussion

According to the case studies in this section, it is enough to demonstrate some key parts of a typical carbon-aware application:

- Provisioning resources for an application could be reduced to optimization problems. The key feature of carbon-aware applications is that they consider emission rates within their optimization model. Different application may have different emission policy expressed either within a constraint of the model or in the goal of the model.

- Carbon-aware applications prefer heterogeneous systems. This feature makes them choose resources from several datacenters whose resources combined could reduce the carbon emission in total.

- Due to the nature of carbon-aware applications, they tend to use resources with less emission rates. Datacenters providing lower emission rates have higher probablity to attract carbon-aware customers.

Chapter 3: Concentrating Renewable Energy

Chapter 2 discussed carbon-aware cloud applications with two case studies. Such applications run inside one or more datacenters by leasing cloud resources from those datacenter owners. To make profit, datacenter owners would encourage applications to use more cloud resources from their datacenters; while keep cost low. Nowadays, renewable energy is more expensive than other alternatives which increases the operational cost to datacenter owners, making them less interest in switching to low-emission energy.

However, our study shows that as long as datacenter owners could attract those carbon-aware customers, they are able to make profit even if renewable energy is slightly more expensive than traditional carbon-heavy energy. Using renewable energy should not only be considered as pure altruism, but also a mean to increasing more cloud resource usage, which leads to more profit. The key insight of our approaches is to concentrate renewable energy to those cloud resources used by carbon-aware customers, instead of evenly distributed among all customers. Under this principle, costly renewable energy could be used effectively to target carbon-aware customers encouraging them to keep using, and/or even use more cloud resources from the datacenter.

In this chapter, we will go through several approaches to concentrate renewable energy to a specific group of customers. First, we try to solve this problem from the architectural perspective. We consider datacenters with on-site renewable energy and focus on the grid-ties — the device most commonly used to integrate renewable energy. The placement of grit-ties will affect the concentration of renewable energy in datacenters. To quantitatively study the renewable energy concentration, we introduce a new metric, the *renewable-powered instance*, that measures the concentration of renewable energy in datacenters. Then, we move our focus onto the software infrastructure level in datacenters. In this approach, we renewable energy is described as *carbon offsets*, which includes not only on-site renewable energy, but renewable energy credit as well. Following our general principle — concentrating renewable energy/carbon offsets only to carbon-aware customers when it leads to profit, we introduce a heuristic algorithm to discover carbon-aware applications based on cloud applications' behavior. Once we locate a carbon-aware application, we profiling its behavior by providing different amount of carbon offsets to find the optimal emission rate which maximize the profit.

3.1 Concentrating Renewable Energy in Grid-Tied Datacenters

Some datacenters reduce their carbon footprint by producing renewable energy on site via solar panels or wind turbines [41, 1, 8]— 7 such renewable-energy datacenters were announced in 2009 [41]. In this section, we study how to concentrate renewable energy in these datacenters with on-site renewable energy generator. We focus on the device most commonly used to integrate on-site renewable energy: *the grid tie.* Grid ties transform electricity from a primary power source (e.g., rooftop solar panels) into electricity with the same frequency and voltage as a secondary power source (e.g., the electric grid). After passing through a grid tie, electricity produced on site can safely replace electricity drawn from the electric grid, reducing the net amount of grid energy used by servers downstream. The grid fills any unmet power needs when on-site sources are insufficient and it accepts excess energy when the on-site sources over produce. Thus, the concentration of renewable energy in grid-tied datacenters depends on 1) the amount of energy produced by the primary source and 2) the number of servers that are downstream to the grid tie. If every server is downstream, the secondary source will probably need to supply some power, diluting the concentration of renewable energy per server. If just one server connects to the grid tie, only that server is guaranteed to have high concentrations of renewable energy. *Our goal was to quantify the impact of grid-tie placement on the concentration of renewable energy.*

3.1.1 Grid-tied Power Delivery

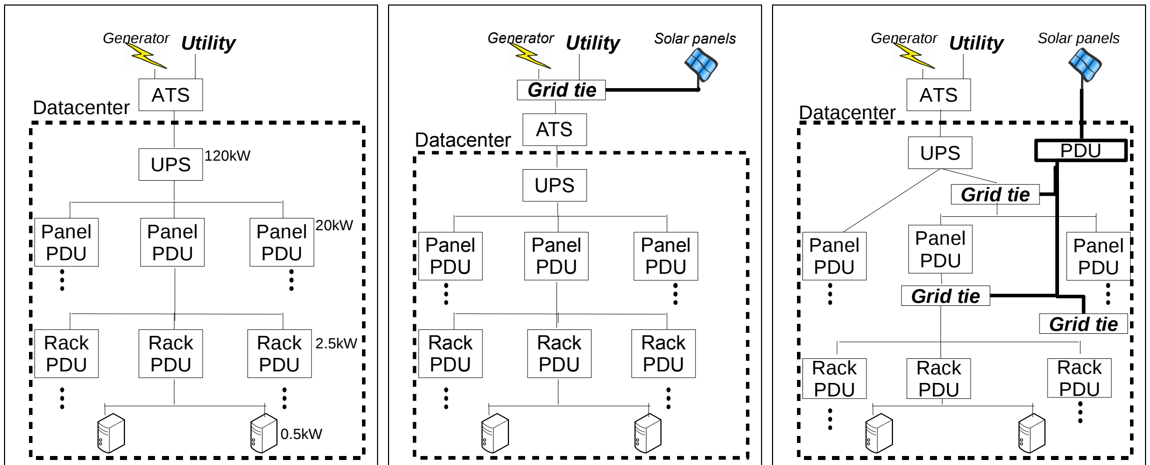
The servers and network switches inside datacenters operate at very low voltages, using much less power than the entire facility. These devices are powered on parallel circuits—i.e., each device has its own receptacle for 120V AC power. These parallel circuits are combined using higher voltage PDUs (208V, 240V, or 480V) that eventually connect to uninterruptible power supplies (UPS) that serve the whole facility [94, 42, 16]. Grid ties can be integrated into any of the above devices. Each integration point places certain requirements on the grid tie during manufacturing, e.g., UPS integration normally requires that grid ties output 480V, 3-phase AC. Most grid-tie manufacturers offer at least 1 product line for each integration point [22], creating a lot of options for the placement of grid ties. Below, we formally describe the grid-tie placement problem and make a case for a new metric to study the concentration of renewable-energy in datacenters.

Power delivery components (P) and independently powered servers (D) in a datacenter can be represented as vertices in a directed acyclic graph, $G = (P \cup D, E)$. Cords in the datacenter are edges (E). Grid-tie placements are represented by a function $gt[e]$ that is 1 only if a grid tie is integrated at edge e; the function is 0 otherwise. Let $e = (v_1 \rightarrow v_2)$, power from the grid tie will first flow downstream to descendants of v_2, i.e., the servers that draw power from v_2's circuit. The amount of renewable energy flowing into an integration point is represented as $r[e]$. The power draw of a vertex v and its descendants is represented by the function $f[v]$. In this model, grid energy used is: $g = f[v_0] - \Sigma_{e \in E} r[e] \cdot gt[e]$.

3.1.2 Renewable Powered Instances

After grid ties transform electricity produced on-site, the concentration of renewable electrons versus dirty electrons can not be measured directly. It is physically impossible to tag renewable electrons. However, prior power engineering research [20] provides a reasonable principle: Electricity from multiple sources is distributed proportionally from sources to loads. Adapted to the datacenter environment, the percentage of renewable energy powering downstream servers equals the percentage of renewable energy used upstream at the grid tie's integration point. More precisely, the percentage of renewable energy powering a datacenter server d_i is:

$$c[d_i] = MAX_j\left(\frac{r[(v_x, v_j)] \cdot gt[v_x, v_j]}{f[v_j]}\right)$$
$$\text{where } d_i \in D \wedge \ d_i \in descendants(v_j)$$



(a) The power delivery subsystem in our department today.

(b) A large grid tie integrated just above the UPS.

(c) Multiple grid ties integrated and controlled.

Figure 3.1: Simplified views of power delivery and grid-tie integration.

We define a k renewable-powered instance (k-RPI) as a server that gets at least $k\%$ of its energy needs from renewable energy. Here, k reflects a minimum level of concentration supplied to green customers and the result is a metric that can be used to compare grid-tie placements.

$$
\begin{aligned}
k - rpi &= \{d_0, d_1, ...d_n\} \\
&\text{where } d_i \in D \wedge \\
\exists v_j, v_x : \ & d_i \in descendants(v_j) \wedge \\
&\frac{r[(v_x, v_j)] \cdot gt[v_x, v_j]}{f[v_j]} > k
\end{aligned}
$$

Figure 3.1(a) shows a portion of the power delivery subsystem for our department's two server rooms (a tier-1 datacenter). There is one UPS operating at 3-phase 480V with a power capacity of 120KW. The UPS delivers power to 7 PDU panels which ultimately deliver 120V-power to 93 rack-level PDUs. These rack-level PDUs power

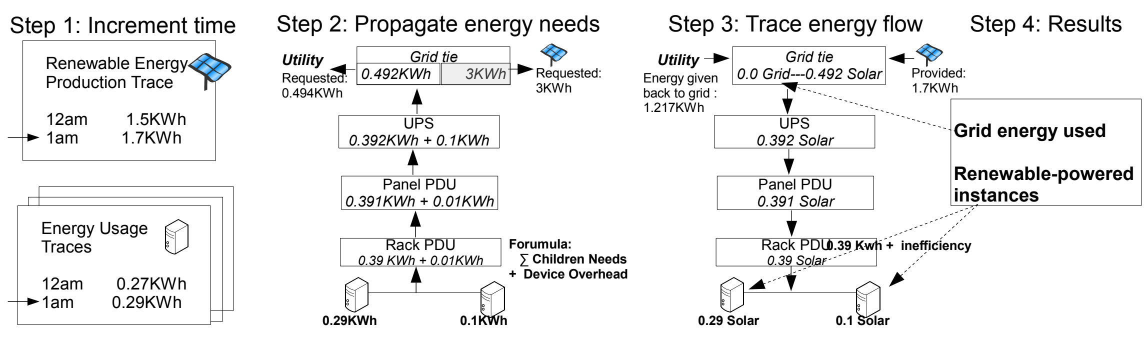


Figure 3.2: Simulating power delivery in a grid-tied datacenter. The far-left upper block shows the hourly energy production in KW-hours of a solar panel system in L.A.,CA. The far-left lower blocks capture energy usage of compute devices (hence, the picture of the server). Labeled blocks in the middle of the figure represent simulated components.

332 servers, networking equipment, and (in 4 cases) other power distribution strips attached to monitors and tower-style servers. On average, there are 32 cores per rack. Figure 3.1(b) shows our subsystem with a grid tie placed between the transfer switch and the utility provider. A real datacenter with on-site renewable energy uses a very similar grid-tie integration strategy [65]. Figure 3.1(c) shows our subsystem with a smart PDU controlling multiple grid ties. Later, we will see that this particular placement performs well.

3.1.3 Trace-Driven Simulation

The power delivery subsystem in a datacenter with on-site renewable energy is affected by the production of renewable energy and the energy usage of servers and networking devices. Three facts will affect the power delivery subsystem in a datacenter with on-site renewable energy: the production of renewable energy, the energy usage of servers and networking devices. Rather than model these complex factors, we acquired traces from renewable energy systems and datacenters. These traces provide realism when we consider the impact of these factors. We were fortunately able to find amount of traces, especially on the renewable energy side. Actually the availability of traces hardly limited our study. The main drawback for our approach is that trace-driven simulation cannot consider the impact of user feedback. For renewable energy, this is not too bad, since users have little effect on the weather. For the energy usage of datacenter devices, users could change their behavior based on the datacenter's green-ness [25]. In future work, we will look to merge our approach with detailed models [76, 83], execution-driven simulation [37] or instruction emulation.

Figure 3.2 shows the simulation of a grid-tied power delivery subsystem for 1 time interval. At the start of each interval, our simulator updates global virtual time. Next, we get the energy needs from all datacenter compute devices at the current virtual time (taken from the trace data). Then the simulator forwards these needs to the power delivery components that the devices are plugged in to. These delivery components add their own energy needs (based on their capacity and the current load) and the simulator forwards the aggregate needs to the next level in the delivery subsystem. This basic propagation model captures the behavior of traditional delivery devices, but the grid tie is unique. As shown in Figure 3.2, the grid tie forwards the aggregate energy need onto its secondary source (the grid) while forwarding a request for its maximum capacity onto its primary source (the solar panels).

The third phase of the simulation begins once energy needs have propagated up to an energy source (i.e., a solar panel system, a wind turbine, or the grid). Then the virtual time is used to get the energy production of each source. The simulator replies to top-level power delivery components with the minimum of energy production or energy needs [20]. To ensure balanced delivery, we assume that the grid can always supply energy needs. Each simulated component subtracts its energy needs from the aggregate needs and delivers the requested amount of energy to the devices plugged into it. Each component also passes a tag representing the proportion of energy supplied by each source. Once again, the grid tie component plays a special role: it ensures that the total amount of energy from its primary and secondary sources does not exceed the amount requested. Our simulated grid-tie component follows three rules:

1. All energy from the primary flows to devices downstream, up to the amount requested.

2. Energy from the secondary source supplies any difference between the primary's production and downstream needs.

3. Excess energy from the primary flows back upstream to the grid.

We built a block diagram simulator, similar to the popular SCICOS framework [24]. However, since we target power delivery subsystems, we optimized our simulator for acyclic directed graphs. We do not implement any function for removing graph cycles, and we only consider discrete states. We need to call each block twice at every time step: one for requesting energy, another for feeding energy from its parents. Each type of block has an associated file for maintaining information about the delivery inefficiency under different levels of utilization.

3.1.4 Study of Renewable-Powered Instances

In our work, we used a trace-driven simulation approach to explore the effect of different grid-tie placements under 2 datacenter power delivery subsystems. The inputs to our simulator were 1) a graph of the power delivery subsystem, 2) a trace of

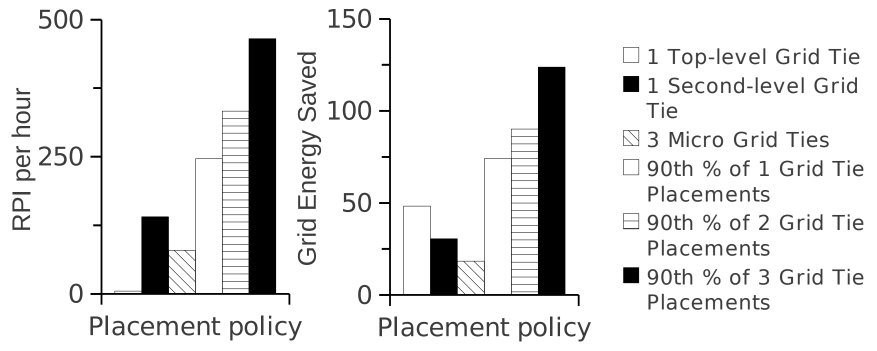
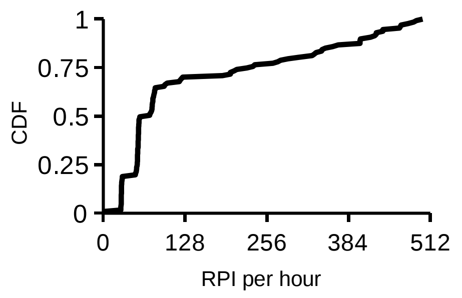


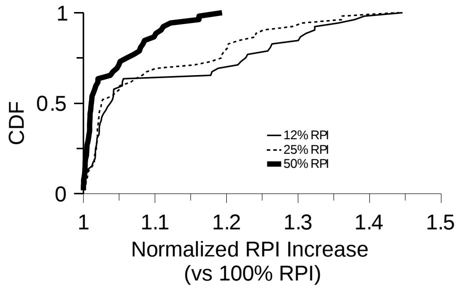
Figure 3.3: Comparing commonly used grid-tie placements to the 90^{th} percentile of RPI producers.

per-server energy needs, and 3) a trace of renewable energy production. Our default datacenter subsystem and trace comes from our department's datacenter which comprises more than 104 rack- and panel-level PDUs. It is possible to integrate a grid tie at each, meaning that even for our small datacenter there are thousands of ways to place multiple grid ties. We used solar and wind renewable-energy traces collected from different places in the U.S.

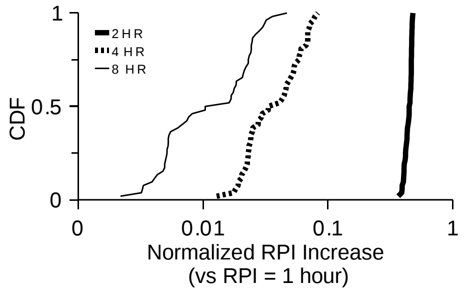
Our department's power delivery subsystem was simulated with 500 randomly selected grid-tie placements. Each placement had at most three grid ties with randomly chosen integration points. After the integration points were selected, we chose the grid tie from [22] that most closely matched the capacity of the integration point (e.g., 240V, 6KW PDU). Data from [22] is used to simulate the grid tie's energy needs (i.e., its inefficiency) as a function of its load. We collected the energy usage of rack-level PDUs in our department from March to June 2010 and looped this trace to get 1 year of virtual time entries for our simulator. For the trace of renewable energy production, we used a 1-year (2004) trace of wind energy production from Cheyenne,



(a) CDF of the 100% 1-hour RPI per hour



(b) RPI = X% renewable power for 1-hour



(c) RPI = 100% renewable power X-hours

Figure 3.4: Performance of 500 randomly selected grid-tie placements under the WY production trace, our department's energy usage trace, and 20% renewable to energy ratio. An RPI indicates that 100% of a compute core's [2] energy for 1 hour came from an on-site renewable source. This our default test setup, unless otherwise mentioned the reader can assume that experiments in this section have these settings.

WY [81], the site of a well-known datacenter with on-site renewable energy [5]. The trace came from the National Renewable Energy Laboratory [78]. We linearly scaled the production trace to produce 20% of the energy used by the datacenter, reflecting an incremental approach to deploying on-site renewable energy.

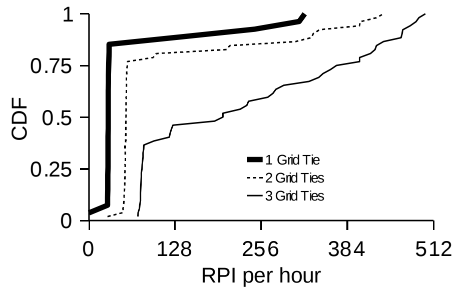
Figure 3.4(a) plots the distribution of 100% renewable-powered instances (100-RPI, or RPI for simplicity) across the tested placements. RPI production varies a lot; some placements allow datacenter managers to track 499 RPI per hour while other placements can't report any RPI. Figure 3.4(a) shows that grid-tie placement *can* affect a datacenter's ability to concentrate renewable energy. In the remainder of this subsection, we examine factors that could vary from one datacenter to another to see if they change this result.

An RPI in our default setup required a grid-tie placement to power a compute core with 100% renewable energy for an hour. Both of these parameters, the percentage of renewable energy and the duration of delivery, could vary from datacenter to datacenter. Figure 3.4(b) shows that decreasing the first parameter improved some placements much more than others. Looking further into the data, we found that the best placements were getting better. When 12% renewable power constituted an RPI, we observed that the 90^{th} percentile of grid-tie placements could produce 539 more RPI per hour than the 10^{th} percentile. This was 1.39 times larger than the difference between the 90^{th} and 10^{th} percentiles when 100% renewable power was required.

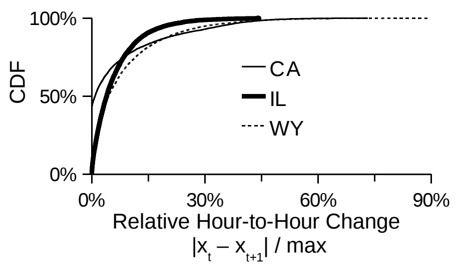
Figure 3.4(c) shows that increasing the duration required for an RPI can significantly reduce the total number of RPI (note the x-axis is log scale). As a result, the absolute difference between grid-tie placements is also reduced. When the required duration of an RPI is 8 hours, the 90^{th} percentile produces only 11 more RPI per hour than the 10^{th} percentile—but these placements only produce 12 RPI per hour total.

Comparison of Common Grid-Tie Placements

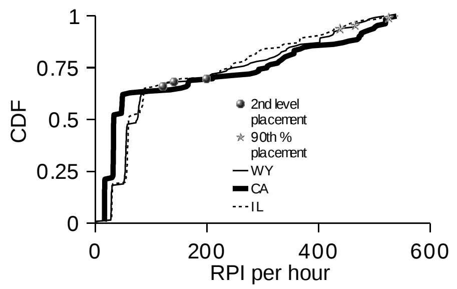
Next, we asked "how well do the grid-tie placements used in practice today perform?" We spoke with 2 datacenter managers who were able to share high-level information about their power delivery structure [65, 60]. Both of these datacenters



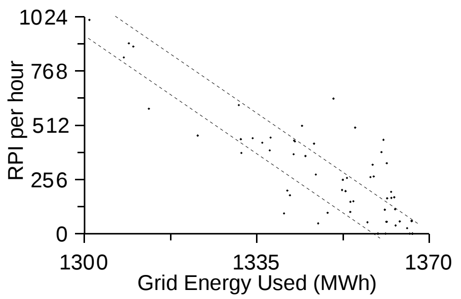
(a) Impact of multiple grid ties



(b) The volatility of renewable-energy sources in Chicago,*IL*, Cheyenne,*WY*, and Rohnert Park,*CA*.



(c) Impact of Geographic Location



(d) Impact of Datacenter Size and Workload

Figure 3.5: The impact of practical design parameters.

integrated only one grid tie near the top of their power delivery hierarchy. One integrated the grid tie above their UPS and ATS systems (see Figure 3.1(b)), meaning that renewable power could support their entire facility. The other integrate the grid tie on the B-side of their datacenter. Here, the grid tie could only power part of the datacenter at a time. In contrast to these 1-grid-tie placement strategies, we also considered a new trend in practice: the use of multiple micro grid ties [40]. Because grid ties consume energy during the process of making primary and secondary sources compatible, and the consumption differs across grid ties with different capacity or under different load [22], placing several grid ties at multiple levels of power delivery subsystem may affect the grid energy used. To study the impact of multiple gird-tie placement, the idea is to integrate renewable energy into the system at easy to install 120V wall outlets rather than larger capacity UPS or PDU integration points.

We applied these grid-tie placements to our department's datacenter and compared the results to the 90^{th} percentile of our randomly selected placements. Note, Figure 3.1(c) shows the 90^{th} percentile placement for the placements with 3 grid ties. Figure 3.3 shows that the placements used in practice performed considerably worse. Below we highlight important results that further confirm the impact of grid-tie placement:

- The naive approach of placing grid ties near the utility provider can significantly decrease RPI production. Placing 1 grid tie wisely at the rack-level or panel-level increased RPI per hour by 1.76X.
- The integration of multiple grid ties improved RPI production by 1.89X while using 60% less grid energy. As shown in Section 3.1.1, simply turning on grid ties consumes a lot of power. When renewable-energy sources produce low amounts of power, big large capacity grid ties can't be turned on—explaining the poor savings from the top-level grid tie. Adding multiple grid ties allows the power delivery subsystem to flexibly use such renewable energy during low production periods.
- Micro grid ties produced 17% of RPI produced by the 90^{th} percentile of 3 grid ties. This was because the micro grid ties were unable to bring in enough

renewable power. A high-capacity grid tie ensures that power produced during windy surges flowed downstream.

- The best placements have significant overhead, almost 9% of the grid energy savings in some cases.

Figure 3.5(a) shows that the use of multiple grid ties increases the disparity between grid-tie placements. The difference between the 90^{th} percentile and the 10^{th} percentile of placements with 3 grid ties was 388 RPI per hour, more than 2.5 times larger than the difference for placements with 1 grid tie. Paired with Figure 3.1(b), these results highlight the importance of grid-tie placement in a new way. Yes, using multiple grid ties can lead to better performance, but it can also increase the opportunity cost from poor placements.

Renewable Energy Production Patterns

We examined renewable energy production in 3 areas used by today's renewable-energy datacenters: Woodstock, IL [65], Rohnert Park, CA [41] and Cheyenne, WY [5]. The CA trace reflects solar-energy production (taken from the Solar Advisor Model [23]), and the IL and WY traces reflect wind-energy production (taken from NREL [81]). Each trace captures hourly snapshots of the monitored energy source's performance (i.e., production divided by capacity) for 2004. The traces were scaled linearly to have the same aggregate energy output.

Not only do these areas reflect different real-world site locations, the production traces offer fundamentally different production patterns. It is well known that solar energy exhibits strong daily cycles and wind energy does not. The auto correlation function with a 24-hour delay (a measure of daily periodicity) was more than 3 times larger (0.88) for the CA trace than for the WY and IL traces (0.09 and 0.27). The WY and CA traces exhibited larger changes from hour-to-hour than the IL trace, shown in Figure 3.5(b). We observe that the largest hour-to-hour changes in CA and WY (72% and 89% respectively) are more than 60% larger than the largest change in the IL trace (44%).

Figure 3.5(c) shows that studied production patterns had only a small affect on the full distribution of grid-tie placements. The median solar placement produced only 1.05X more RPI than the median wind placements, between the wind sites, the distributions were essentially the same. These results suggest that datacenter managers in a wide range of areas can expect similar returns on investments in renewable energy. Figure 3.5(c) also explores the performance of two specific placements, the 2^{nd} level placement approach described earlier and the 90^{th} percentile placement in Cheyenne, WY. We were surprised to see the the performance of the individual placements varied a lot, producing 1.5X and 1.8X more RPI in the solar study. When we investigated this result, we found that these placements both place a grid tie at the largest panel-level PDU which happens to have energy consumption patterns that closely match the daily peaks of solar energy production. If our simulated solar panel system produced just 10% more energy, the placements would send a substantial amount of renewable energy upstream and would be less effective relative to their performance under the wind sites. This results shows that the performance of individual grid-tie placements can vary from location to location.

Next, we examined the impact of amount of renewable energy produced on site. We scaled our production traces to produce 5%, 20% (default) and 100% of the energy used by the datacenter's energy consumption traces. We found that the tail of the distribution became much heavier as we increased the amount of renewable energy produced on site. The difference between the 90th percentile and 10th percentile was 1.56 times larger when the amount of on-site production was equal to the on-site consumption compared to the default case. Comparatively, it was only 6% smaller when the on-site production was 5% of aggregate energy needs.

Datacenter Size and Workload

We concluded our study by examining grid tie placement on a larger datacenter. We monitored the energy usage and mapped the power delivery subsystem for a datacenter serving our entire university (one of the largest university's in the nation). The workload at this datacenter differed from the mostly research workloads in our department, running more enterprise-oriented applications, like SAP and BlackBoard. Further, this datacenter uses virtualization and other energy efficiency techniques. We compared the hour-to-hour volatility of the energy usage between these datacenters and found that the average hour-to-hour change in the university-wide datacenter was 10 times more than in our department datacenter which we attribute to fluctuating workload and higher energy efficiency. The median hourly change of university-wide PDUs was 15% of its peak consumption.

Figure 3.5(d) plots the grid energy used against the RPI for the university-wide datacenter. This result provides further confirmation of our hypothesis that grid-tie placements can significantly affect the ability to concentrate renewable energy. We once again observe that the worst placements essentially eliminate the ability to track when servers are powered by renewable energy (i.e., RPI). In future work, we plan to study the economic value of the RPI metric.

3.1.5 Discussion

According to our simulation-based study, proper grid-tie placement can increase the degree of renewable energy concentration. Carbon-aware customers could be able to use more renewable energy from the datacenter. However, there are some restrictions about this appraoch:

- It requires the datacenter has on-site renewable energy generators. This will introduce additional cost to the datacenter owners.
- Placing grid-ties requires architectural level changes. Once grid ties are placed, they will be rarely moved; while cloud resources may be leased frequently.
- This study only considers how to concentrate renewable energy. It did not consider any applicable profit model to help datacenter owners how to use those renewable-powered instances.

3.2 Adaptive Green Hosting

As we discussed in Chapter 1, there is a niche market focusing carbon-aware customers by providing green computing resources. Some datacenter owners provide *green hosting*[1] service which leases cloud resources powered by clean renewable energy while maintaining low prices.

Green hosting firms are targeting a small but growing market, cloud application owners that want show their commitment to the environment. Most people worldwide (83%) say that they prefer green products when they do not cost more than non-green alternatives [55], perhaps reflecting conspicuous altruism [54]. Similar results show

[1] The term "green Web host", instead of green datacenter, is widely used in popular press and on business websites [79, 1, 56]. In this section, we follow this precedent.

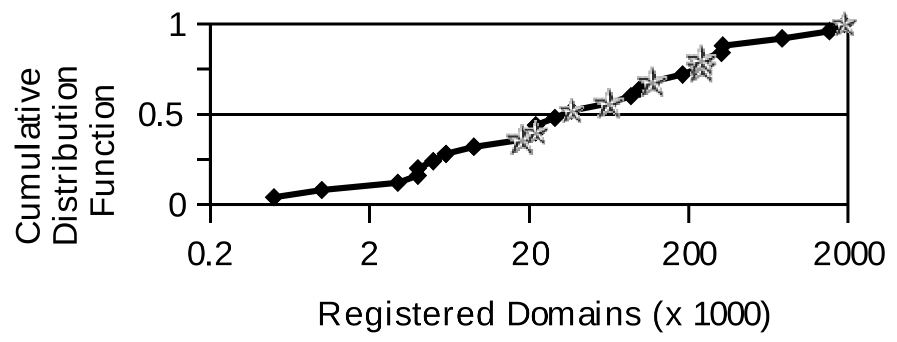


Figure 3.6: A niche market for green hosting. Each point represents 1 Web host. The X-axis is the number of A-type DNS records registered to the host. Stars indicate green hosts.

that CIOs (61%) and system managers (71%) are willing to support green hosts if prices, response times, and throughput are the same [82, 25].

Figure 3.6 provides evidence of the growth. We plot registrations on the domain name services (DNS) of Web hosts, a rough but widely used metric to size Web hosts. Hosts with more authoratative (A-type) DNS records likely support more applications. Using Domain Tools [36], we counted the DNS records of 200 Web hosts returned from online searches, plotting the 25 largest. In this group, there were 8 hosts that mentioned clean energy investments on their public Web pages (green hosts) and 17 traditional hosts that did not. We also controlled for price and hosting features. Each host offered hosting plans below $5, a 99.9% uptime guarantee, and unlimited network data transfer. Most green hosts in our study were above the median in terms of registered domains with 2.9 times more A-type records than traditional hosts on average.

However, due to the intermittent nature of renewable energy production, it usually costs more than traditional dirty energy. Green hosts must host more applications than traditional hosts to profit from their investment. We propose *adaptive green hosting*, a software infrastructure layer component for datacenter owners to attract carbon-aware customers by efficiently investing clean, renewable energy. It is datacenter owners' response to geographically distributed carbon-aware applications. As we discussed in Chapter 2, such applications meet their cost, SLA, and carbon footprint goals by routing their workload to hosts that offer either 1) low carbon footprints, 2) high performance, or 3) a little bit of both. Our key insight is that a green host can entice an application to route workload to it by providing more carbon offsets. As mentioned in the beginning of this chapter, we use the term *carbon offset* to represent a unit of clean energy that can replace a unit of dirty energy, both measured in Joules. Carbon offsets can be produced by on-site solar panels, power received from local wind farms, or renewable energy credits (RECs) purchased via energy markets. In adaptive green hosting, Web hosts set the ratio of carbon offsets to dirty energy, henceforth the *offset ratio*, by observing their profit from each hosted application under various settings over time. Where geographically distributed cloud applications look for hosts with good offsetting policies, adaptive green hosting sets policies to entice cloud application owners to use them.

3.2.1 Making the Case for an Adaptive Approach

Adaptive green hosting contrasts with the approach most widely used in practice today, fixed offset ratios. In choosing a fixed offset ratio, today's green hosts try to meet the carbon footprint goals of their hosted applications. However, meeting this threshold does not ensure that a green host will receive an application's workload. Instead, an application may route its workload across multiple hosts, mixing resources

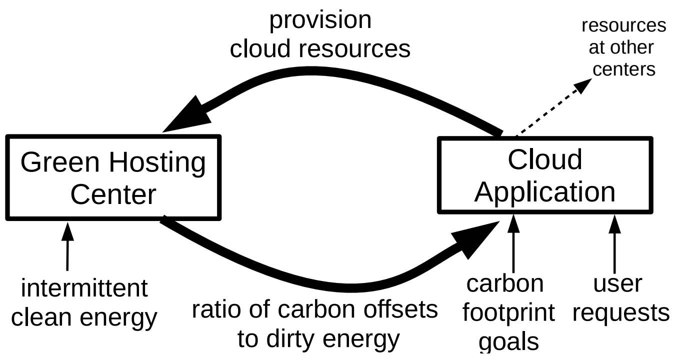

provision cloud resources
resources at other centers
Green Hosting Center
Cloud Application
intermittent clean energy
ratio of carbon offsets to dirty energy
carbon footprint goals
user requests

Figure 3.7: Adaptive green hosting for cloud applications that lease resources on demand.

that differ in performance and offset ratios. This latter approach exploits fungible carbon offsets. The application needs only ensure that the weighted sum of their carbon footprint across all hosts meets their goals.

Adaptive green hosting introduces a new control loop based on carbon offset ratios (shown in Figure 3.7). Hosts adapt their offset ratios for each hosted application in response to changes in the availability of carbon offsets and request arrival patterns. We define a *carbon offset policy* as a vector where each element indicates the offset ratio assigned to each hosted application. This section makes the case for an adaptive approach by showing that fixed policies yield below optimal profits even when carbon offsets are always available at a fixed price and hosted applications have fixed carbon-footprint goals.

Motivating Example

Consider Ecosia [39], a simple application that provides a wrapper to Bing's search APIs and uses ad revenue to 1) offset Bing's estimated footprint and 2) invest in a rainforest protection program. Rather than spending its ad revenue on carbon offsets for the servers that host its homepage, CSS style sheets, and CGI scripts, Ecosia uses green hosts, bundling the costs of carbon offsets with hosting expenses. Ecosia commits to a carbon neutral footprint for its servers [39], i.e. 100% offset ratio. That is, Ecosia must be able to attribute 1 carbon offset for every joule of dirty energy used to power its servers. Every month these servers support more than 15 million unique searches that must complete quickly or else Ecosia will lose users [4].

For this example, we assume that Ecosia can send search requests that originate in the East Coast of the US to a Web host in either 1) the Eastern US, 2) the Western US, or 3) Europe. This setup mimics prior work [66, 32]. The hosts differ only in their network latency and carbon offsets per joule. The eastern host has the lowest network latency (41ms round trip on average), then the western host (80ms), and finally the European host (121ms). Each host leases cloud instances that can service a request in 1.6ms, supporting up to 600 requests per second (RPS). However, successful requests must complete within 150ms, including network latency, queuing delay, and service time. The expected successful requests from each datacenter is shown below, using an modified M/M/1 queuing model [59].

$$\text{Eastern US Host} \qquad v_0 = \frac{(0.150 - 0.041)}{\frac{1}{(600-\lambda_0)}} \tag{3.1}$$

$$\text{Western US Host} \qquad v_1 = \frac{(0.150 - 0.080)}{\frac{1}{(600-\lambda_0)}} \tag{3.2}$$

$$\text{European Host} \qquad v_2 = \frac{(0.150 - 0.121)}{\frac{1}{(600-\lambda_0)}} \tag{3.3}$$

Here, λ_0 reflects the request arrival rate at time 0. The eastern host offers no carbon offsets, the western host is carbon neutral, and finally the European host buys 2 offsets for every joule it uses. In other words, the hosts have offset ratios of 0%, 100%, and 200% respectively.

Ecosia wants to use as few cloud instances as possible while ensuring 1) all arriving requests complete successfully and 2) carbon footprint goals are met. Cloud instances are leased hourly. We assume that at every 1-hour interval t, Ecosia knows its request arrival rate for that interval, e.g., $\lambda_t = 120$ requests per second. With the request arrival rate, we can compute how many requests each instance can complete successfully (i.e., $v_0 = 51$,$v_1 = 38$,$v_2 = 16$ under $\lambda_t = 120$ RPS). Knowing the offset ratio for each instance (i.e., $c_0 = 0\%, c_1 = 100\%, c_2 = 200\%$) and Ecosia's goal of being carbon neutral ($C = 100\%$), we can compute the Ecosia's optimal workload distribution, i.e., the vector $X =$ ¡$x_0,\ x_1, \cdots, x_i$¿ where each element reflects how many instances (an integer) Ecosia leases from each host i. The formal optimization model is:

$$\text{Minimize } \sum_{i=0}^{n} x_i^{(t)} \tag{3.4}$$

$$\text{Subject to } \frac{\sum_{i=0}^{n} E_i c_i^{(t)} x_i^{(t)}}{\sum E_i x_i^{(t)}} \geq C \tag{3.5}$$

$$\text{and } \sum_{i=0}^{n} v_i x_i^{(t)} \geq \lambda \tag{3.6}$$

$$\text{and } \forall_i (x_i \in \mathbb{Z}) \tag{3.7}$$

The goal is to minimize the total number of instances used. The first constraint keeps Ecosia's servers within a target carbon footprint (C). To be carbon neutral, Ecosia would set $C = 0$. Assuming green hosts and traditional hosts differ only in their offset ratio, we uniformly set the energy per instance coefficient (E_i) to 100wH.

The second constraint requires enough instances to process incoming requests (λ_t) within SLA.

Integer programming solvers can find near optimal workload distributions for Ecosia [66, 72, 107]. We used LP solve, an open source solver commonly bundled with Linux platforms [73]. Under 120 RPS, Ecosia would use 4 instances from the host in western US only. Even though the host in eastern US can successfully complete 1.3X more requests per instance, the lack of carbon offsets forces Ecosia to use other hosts.

Under adaptive green hosting, the eastern host could buy carbon offsets specifically to attract Ecosia's workload. The carbon-offset elasticity (η) captures a host's workload as a function of carbon offsets assigned (ζ) to a target application. The carbon-offset elasticity tells us if a host can increase its workload by giving a target application more offsets per joule of dirty energy. These offsets can be bought as renewable energy credits, transferred from another application, or pulled from on-site sources. Because energy is fungible, this is an accounting problem. Below, we show the optimization formula for carbon offset elasticity for a single application. Equation 3.9 projects Equation 3.5 to a single host that considers the marginal gain by changing its offset ratio (Equation 3.10).

$$\eta_j(\zeta) = x_j : \text{ Minimize } \sum_{i}^{n} x_i^{(t)} \tag{3.8}$$

$$\text{Subject to } \frac{\sum_{i=1}^{n} c^{(t)} x_i^{(t)} + \zeta x_0^{(t)}}{\sum x_i^{(t)}} \geq C \tag{3.9}$$

$$\text{and } \sum_{i=0}^{n} v_i x_i^{(t)} \geq \lambda \tag{3.10}$$

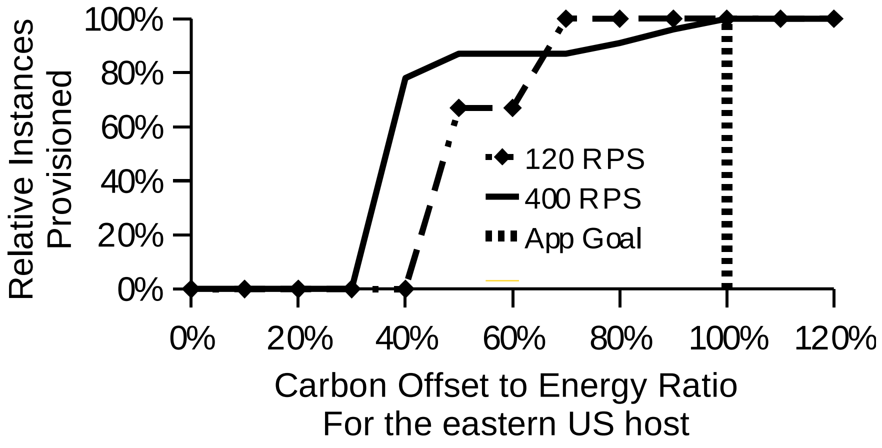


Figure 3.8: Carbon-offset elasticity for the eastern US host. The y-axis shows instances provisioned on the host relative to the maximum setting, i.e., $\frac{\eta_{east}(X)-\eta_{east}(K)}{\eta_{east}(K)}$ where K maximizes η_{east}. Ecosia routes requests differently across offset ratios (x-axis). Under 120 requests per second (RPS), $\eta_{east}(K)$ equals 3 instances. Under 400 RPS, it equals 23 instances.

For N discrete settings of ζ, we can compute a host's carbon offset elasticity for a model-driven application by solving N integer programming problems. *We use this key insight to assess the yield of clean energy investments for a host.*

Figure 3.8 shows the carbon-offset elasticity for the eastern host. The result highlights a unique aspect of clean energy: it is fungible. Even though Ecosia managers want their application to be carbon neutral, they will lease instances from a host that offsets less than 100% of its carbon footprint if other hosts offset more than 100%. In this example, the eastern host benefited. Under 120 RPS, if the eastern US host were to offset just 50% of its carbon footprint, the best workload distribution used only eastern US and European instances. If the eastern US host were to offset 70% of its carbon footprint, the best workload distribution used 1 European and 3 eastern US instances, matching the the number of instances used if the host were to offset 100% (carbon neutral).

The carbon elasticity changes when Ecosia's request arrival rate rises to 400 RPS. Under 70% carbon offset ratio, European instances detracted 13% of the workload that would be sent to the eastern US host if it were carbon neutral. In fact, under 400 RPS, the eastern host leases the same number of instances under a 50% offset ratio as it does at the 70% offset ratio. This shows that a static carbon offset policy chosen under 1 request arrival rate can be below optimal when the request rate changes. Note, this finding does not require that Ecosia managers change their carbon footprint goals or relax their SLA, nor does it require that carbon offsets become more or less available. Also, we observe that the elasticity function grew slowly after 40% offset, raising the question, "does a 15% increase in leased instances justify a 60% increase in the offset ratio?" We address this question Section 3.2.2.

Generalizing the Example

The relative throughput and offset ratios of the hosts in our example capture a practical region of the workload distribution problem for carbon aware applications. The general problem is an integer programming problem; each application assigns an integer ($ip_i \in \mathbb{Z}$) to n-tuples (v_i, c_i) reflecting the instances leased from each host. The ideal solution is not limited by the integer requirement and finds a solution equal to the linear programming solution ($lp_i \in \mathbb{R}$). We constrain this space of problems with the following assumption: an application will consider only 1 host that doesn't meet its carbon footprint goals, the best performing host. Our assumption builds from the intuition that workload distribution involves some management costs that will deter managers from choosing poor performing hosts that offer too few offsets to meet an

Conditions	*Hosts chosen*
$c_{BP} \geq C$	East
$\forall_i lp_i(\lambda) \in Z$ and	East, West, or Euro
$\sum v_i \cdot [lp_i(\lambda) - \lfloor(lp_i(\lambda))\rfloor] \leq 2v_{sp}$	East, West, or Euro
$\sum v_i \cdot [lp_i(\lambda) - \lfloor(lp_i(\lambda))\rfloor] > 2v_{west}$ **and** $\sum v_i \cdot [lp_i(\lambda) - \lfloor(lp_i(\lambda))\rfloor] \leq v_{east}$	East, West Euro, $Euro_\infty$
$\sum v_i \cdot [lp_i(\lambda) - \lfloor(lp_i(\lambda))\rfloor] > 2 * v_{east}$ **and** $\sum v_i \cdot [lp_i(\lambda) - \lfloor(lp_i(\lambda))\rfloor] \leq v_{east} + v_{euro}$	East, West Euro, $Euro_\infty$

Table 3.1: A summary of all outcomes for the workload distribution found via integer programming solution for carbon-capped and performance-oriented applications.

application's goals. We call applications that follow this assumption performance oriented.

We claim that any carbon-capped and performance-oriented application will lease instances from only 1) its best performing host, 2) its best performing host that meets carbon footprint goals (i.e., second best performing host), or 3) the host offers an offset ratio that exceeds the application's footprint goals and combines with the best performing host to yield highest performance per instance achievable while meeting footprint goals (exploiting fungible offsets). These properties correspond to the eastern US, western US, and European hosts in the Ecosia example. Changing the absolute throughput and offset ratios of an application's hosts will change the proportions with which each host is selected. However, our claim is that any host used by a carbon capped application will have (in the limit) at least one of the 3 properties above.

We prove this claim by considering all possible outcomes of the optimization model. Table 3.1 provides a summary of our proof.

Outcome #1: The best performing host offers an offset ratio that exceeds the application's carbon footprint goals. The application uses instances from only the eastern US host.

Outcome #2: The linear programming solution returns only integer values. With only 2 constraints, an n-host linear programming solution chooses between only 3 hosts [97]. The application picks instances from either only the western US host or some linear combination of the eastern US and European hosts, whichever provides the best performance per instance. The outcomes here are restricted by our prior assumption that each application considers at most 1 dirty host. We leave to future work an extension of this analysis for applications that can use more than 1 dirty host. For such applications, any mix of dirty and green hosts could be the most efficient, which would make computing the η function more complex.

Outcome #3: The linear programming solution uses fractional instances to process fewer than $2v_{west}$ requests. Here, the integer programming solution replaces the fractional instances with whole instances. The western US host represents the most efficient way to do this, since, by definition, it offers the greatest performance among hosts that meet carbon footprint goals.

Outcome #4: The linear programming solution uses fractional instances to process fewer than v_{east} requests. Here, we rely on the performance-oriented assumption. The application either provisions (more than 2) instances from only the western US host, or it mixes instances with the eastern US host and some other host that exceeds its footprint goals. The application must use either the eastern US host or the western US host because no other host offers fewer offsets than the western US host and exceeds its throughput. As the Europoean host's offset ratio goes to infinity, we can show that it becomes the host that the eastern US host is combined with. Thus, we denote it as $euro_{\infty}$ in Table 3.1.

Outcome #5: The linear programming solution uses fractional instances to process fewer than $v_{east} + v_{euro}$ requests. This outcome combines instances from Outcome #3 and #4. Finally, we note that the linear programming solution would not process more fractional requests than $v_{east} + v_{euro}$.

3.2.2 Adapting to Real Workloads

Section 3.2.1 described a cloud application that divided user requests among competing hosts to 1) be carbon neutral and 2) keep its costs low. Hosts received a portion of the application's requests, depending on their cost to throughput ratio, carbon footprint, and the rate at which user requests arrived. This example showed that, as request rates change over time, green hosts that use fixed offset ratios will sometimes lower their profit by buying too many (spending more than needed) or too few offsets (losing customers).

This section shows that green hosts can increase profit derived from an application by eschewing fixed offset ratios in favor of an adaptive approach. Prior research on adapting to workload changes has focused on how applications should provision instances to maximize throughput [9], minimize costs [45, 72], and meet carbon goals [66, 72, 107]. In this section, we focus on how *hosts* should set their offset ratio (e.g., by buying RECs) to maximize their profit for an application. Like prior work, this function depends on the application's request rate, cost models, and carbon footprint goals. However, unlike prior work, this function also depends on the performance and offset ratios of other hosts.

We revisit our example application from Section 3.2.1. This time, we use a trace from a real enterprise application to capture changing request rates. For each 1-hour window in the traces, we compute carbon offset ratios that maximize profit for the eastern, western, and European hosts. We study 1) how many times the best carbon offset ratio changes, 2) how quickly it changes, and 3) how much it changes. Our results prompted us to create a reactive approach that adapts the offset ratio based on recent history. We begin by presenting a formal profit model for green hosting.

Profit Model

Datacenters adopt a cloud computing model earn money by leasing virtual resources over a fixed period of time [16]. A leasable resource is called an instance. Datacenters profit when they earn more money per leased resource than they spend buying, maintaining, and powering them (captured in Equation 3.11).

$$P = I \cdot p \cdot R - \frac{StartupCosts}{T} \tag{3.11}$$

In the above equation, profit P is a function of instances leased (I), revenue per instance (R), the percentage of revenue turned into profit considering only operational costs (p), and amortized startup costs (where T captures the datacenter's expected lifetime). We assume $I \geq 1$. In most places, clean energy costs more than dirty energy, so green datacenters will have higher operational costs. They must lease more instances to profit from this investment.

$$P(c) = \eta(c) \cdot p \cdot R - c \cdot E \cdot cost_{co2e} \cdot S - \frac{StartupCosts}{T} \tag{3.12}$$

$$P(c) = \eta(c) \cdot p \cdot R - c \cdot E \cdot cost_{co2e} \cdot \lceil \frac{\eta(c)}{S} \rceil S - \frac{StartupCosts}{T} \tag{3.13}$$

Equation 3.12 adds the cost of carbon offsets ($cost_{co2e}$), energy per instance (E), the granularity of energy data (measured in instances) (S), and the ratio of carbon offsets to joules (c). These factors make green datacenters less profitable than traditional datacenters. The equation also shows the effect of carbon offset elasticity ($\eta(c)$) in increasing the amount of instances leased. Green hosts can profit by investing in clean energy only when the carbon offset elasticity leads to increased revenue. Equation 3.13 shows the full profit model when $\eta(c)$ can exceed S.

In practice, datacenters invest in clean energy with caution, trying to keep the risk of losing money low. Here, we formalize a risk aware approach commonly used in practice [65, 60]. The idea is to cap how much money is invested in clean energy so that a small increase in leased instances yields profit.

Low Risk Green Hosting: *The maximum ratio of carbon offsets to dirty energy (c_{max}) is capped, such that $c_{max} \leq \frac{pR}{E \cdot cost_{co2e} \cdot |S|}$. Where S is the set of leasable instances receiving the offsets. Plugging c_{max} into Equation 3.12, we see that it allows a datacenter to recoup costs when increasing the offset ratio from 0 to c_{max} yields only 1 leased instance (the worst case).*

Theorem: A datacenter that invests with the above low-risk approach should choose the smallest c that maximizes $\eta(c)$ in order to maximize profit. Here, we provide a short proof. First, we observe that a datacenter's costs are linear in c, provided $E{>}0$ and $cost_{co2e}{>}0$. If $\eta(c+\epsilon) = \eta(c)$, then costs under $c+\epsilon$ would exceed costs under c, meaning lower total profit. Thus, the smallest c is a necessary condition. Second, we

Public Data		
variable	value	source
R	\$0.085	Amazon EC2 [2]
p	4%	Amazon's EBITDA [98]
$cost_{co2e}$	\$0.0045	Renewable energy credits online [52]
Local Tests		
variable	value	source
E	23Kj	ARM Marvel processor + SSD
S	32	Tripp Lite PDU with power display

Table 3.2: Values used to estimate c_{max} for this study.

prove by contradiction that η must be maximized.

$$Hypothesis : Assume P(c_1) > P(c_2) \text{ where } \eta(c_1) < \eta(c_2) \tag{3.14}$$

$$WLOG : StartupCosts = 0 \tag{3.15}$$

$$Substitution : P(c_1) = \eta(c_1)pR - c_1 E cost_{co2e}|S| \tag{3.16}$$

$$Substitution : P(c_2) = \eta(c_2)pR - c_2 E cost_{co2e}|S| \tag{3.17}$$

$$WLOG : Assume c_1 = 0 \tag{3.18}$$

$$Substitution : \eta(0)pR > \eta(c_2)pR - c_2 E cost_{co2e}|S| \tag{3.19}$$

$$Algebra : \frac{c_2 E cost_{co2e}|S|}{pR} > \eta(c_2) - \eta(0) \tag{3.20}$$

$$WLOG : Assume \eta(c_2) - \eta(0) = 1 \tag{3.21}$$

$$WLOG : Assume c_2 = c_{max} \text{ i.e., as large as possible} \tag{3.22}$$

$$Contradiction : \frac{|S|}{|S|} > 1 \tag{3.23}$$

Finally, we used both public data and local tests to calibrate a realistic c_{max}. Table 3.2 shows inputs to our profit model and their source. Our local setup uses a small cluster of ARM processor devices with attached SSD storage. These devices host Apache on Linux, supporting up to 600 requests per second throughput. These results match findings from prior work [70, 83]. The peak power from our ARM nodes is 5.5W; multiplying by 3,600 seconds provides our value for the hourly energy usage of an instance. We also consider a PUE of 1.2. Most (81%) power distribution units (PDU) used at the rack level in today's datacenter include LCD displays and network access for energy data [93]. Our PDU can support 32 instances. Note, the PDU is a good level to assign carbon offsets since energy data is easy to acquire. Assigning carbon offsets at higher levels in the power delivery system increase the size of S, diluting the amount of carbon offsets that can be purchased with low risk [33]. Filling these values into our model, we set $c_{max} = 300\%$ for all studies in the remainder of this section.

Trace-driven Study

We used empirical traces of request rates and carbon prices to study the most profitable carbon offset ratios for green hosts over time. Recall, in Section 3.2.1, we computed η for the eastern host using a constant request rate and the default offset ratios of the western US and European hosts. In this section, we compute η for T timestamped request rates and offset ratios. Assuming low risk investing, the output reduces to a vector of T carbon offset ratios for each host, where the t^{th} setting reflects the smallest ratio c_t that maximizes $\eta_t(c_t)$ given the request rate λ_t. Our final assessment of profit uses our model to combine results from all T time steps.

Figure 3.9 shows two normalized request rate traces taken from an HP cloud application used across the world [92]. These traces cover approx. 8 days and capture diurnal patterns in the request rate. Both traces were normalized to produce about 1.5 million requests per day (about 175 RPS). They differ in the distribution of request rates within a day. The top trace matches the distribution of all arriving requests. Its 99^{th} percentile of request rates is 1.5X larger than the 99^{th} percentile of an exponential

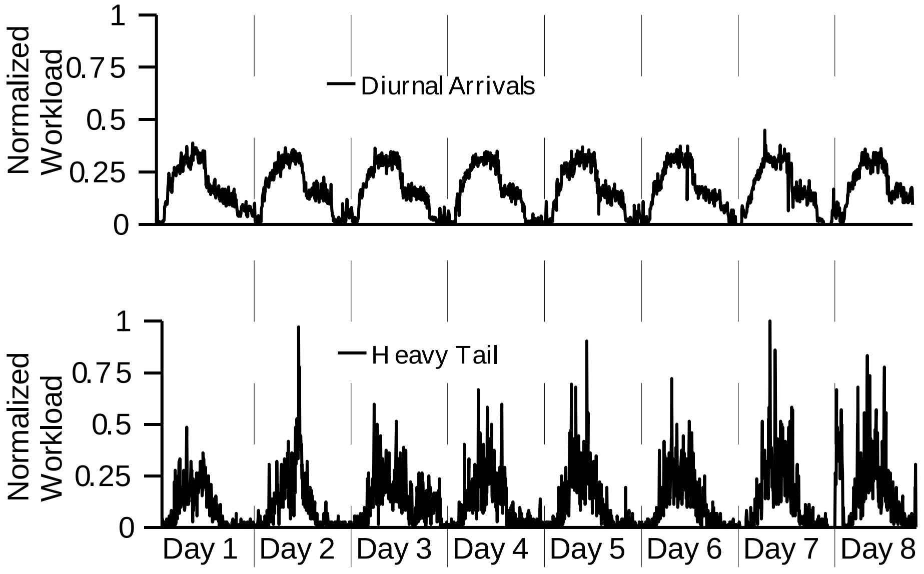


Figure 3.9: Request rates for a modern enterprise application, codenamed VDR [92]. VDR is used in six continents. The plots show requests rates at 2 servers hosted in the Americas. The first plot compiles arriving requests for both servers, capturing diurnal patterns. The second plot shows request rates for a request type with fast response times, likely static content. In the second plot, requests arrive according to a heavy tail.

distribution with the same mean. In other words, the top trace has a tail that is only slightly heavier than an exponential distribution. The bottom trace captures the arriving requests for 1 request type. The 99^{th} percentile of request rates in this trace is 5X larger than the 99^{th} percentile of a normal distribution with the same mean. In other words, the bottom trace has a tail that is much "heavier" than an exponential distribution. Such heavy tails are a well studied in cloud applications [29, 11].

We also studied the effect of changing carbon prices by discounting c_{max} and default offset ratios. We used a trace of the daily market price for carbon offsets from iPath Global Carbon [6]. Our trace ranged from Feb. 8, 2012 through Feb. 14, 2012. The resulting daily, relative prices were 1.08, 1.08, 1.03, 0.98, 0.97, 0.94, and 1. Market prices often track wholesale prices well.

Study Results: We used the iterative method described in Section 3.2.1 to compute carbon offset elasticities for each host, workload, and time step. We chose 31 discrete values for the offset ratio, using multiples of 10% from 0 to 300%. For every 1-hour time step, we used the request rate (λ_t) from either the diurnal or heavy tail traces above to compute how many instances a host would provision if it set its offset ratio to one of the above discrete values. We assume that the other hosts keep their default offset ratio.

Table 3.3 shows how the best carbon offset ratio changed over time under 1) the diurnal workload with fixed carbon prices, 2) the heavy tail workload with fixed carbon prices, and 3) the heavy tail workload with changing carbon prices. In total we computed 1,674 offset elasticities (3 hosts x 3 workloads x 186 hours).

To maximize profit across each studied workload, every host needed to use at least 3 different offset-ratio settings. The hosts used fewer settings (below 4) under the diurnal workload than under the heavy tail traces. We explain these results by highlighting a key aspect of the integer programming (IP) outcomes outlined in Section 3.2.1: Linear programming (outcomes #1 and 2) provide the best solution modulo the request rate. When the request rate is larger than $v_{east} + v_{euro}$, a host should set its offset ratio to maximize its usage under the linear programming solution. For the eastern and western US hosts, this setting is very close to the application's carbon footprint goal. For the European host, this setting is the smallest setting that ensures the following $k * v_{east} + v_{euro} > v_{west}$ where k in $\mathbb{R}$ is the number of eastern US instances sponsored by the European host's fungible offsets. However, when the request rate falls below $v_{east} + v_{euro}$ (outcomes #3–5), the best settings change depending on the IP solution.

In our example, $v_{east} + v_{euro}$ equals 67 RPS and the average arrival rate is 175 RPS. The distribution of request rates in the diurnal workload is close to exponential, meaning the median request rate is close to the average rate. Indeed only 18% of the 1-hour intervals under the diurnal had request rate below 67 RPS. The offset ratio found under the linear programming solution was chosen of 90% of the time for all hosts. Heavy tail distributions do not share this property. Instead, short workload bursts make the average rate larger than the median. Despite having the same arrival rate, the heavy tail distribution shows request rates below 67 RPS 45% of the time. More generally, we can not claim that all heavy tail workloads on all cloud platforms will include some intervals where $\lambda_t < v_{east} + v_{euro}$. However, for a given average arrival rate, a high variance, heavy tail distribution is more likely than an exponential distribution to include such intervals.

Second, we observe that offset ratios change slowly. In particular, we observe several long contiguous periods under the second most frequent policy, even under the diurnal workload. Several last longer than 4 hours. We note that this correlated behavior is well explained by low request rates 1) at night and 2) between bursty periods. The average period of contiguity rounds up to 2 hours in all but 1 of the study traces. Finally, we also observe that the absolute distance between the second most frequent and most frequent ratio are far apart, simply setting the offset ratio to the larger of the two can waste a lot of money.

A Reactive Approach

Since our trace-driven approach revealed that the best offset ratio held for long contiguous periods, we implemented a reactive approach to set the carbon offset ratio. We assume that applications tell each datacenter what their ideal offset ratio was for the previous hour. Given that the application can monitor its request arrival rate, it can compute this offset directly using the approach described in Section 3.2.1.

Our reactive approach considers the history of an application's ideal offset ratio. When the ideal ratios over the last 2 hours match, we change the offset ratio to the matching value. Otherwise, we assign the ratio to the statistical mode. The latter works well under diurnal workloads where the most frequent ratio occurs 97% of the time. The former helps with heavy tailed workloads where the ratio changes for several hours at time.

Our full approach also exploits heavy tailed contiguous periods in the offset ratio. We scan the history of results for patterns indicating that a contiguous period of length l has a large probability of leading to a period of length $l + k$. If such patterns are found, our reactive policy returns to the mode after the $l + k$ interval.

Metric: Number of Ratios Chosen ($|\bar{c}| : c \in \bar{c}$ iff $\exists t\ \forall k\ P_t(c) \geq P_t(k)$)
What to look for: Numbers greater than 1 suggest that fixed policies yield below optimal profit.

Workload	*Eastern*	*Western*	*European*
Diurnal	4 settings	3	3
Heavy tail	5	4	5
Heavy tail and market carbon prices	7	8	5

Metric: Expected Contiguity of 2nd Most Frequent Setting (Ex($|L|$: $\forall$ l $\in$ L $C_l = k \wedge$ (l+1) $\in$ L)
What to look for: Large numbers suggest carbon offset ratios are stable.

Workload	*Eastern*	*Western*	*European*
Diurnal	1.60 hours	1.75	1.23
Heavy tail	1.65	3.00	4.05
Heavy tail and market carbon prices	2.05	2.75	8.86

Metric: Absolute distance from 1st
What to look for: Large numbers reflect the magnitude of profits lost by static hosts. Minimum value is 10% and maximum is 300%.

Workload	*Eastern*	*Western*	*European*
Diurnal	100%	100%	200%
Heavy tail	100%	50%	190%
Heavy tail and market carbon prices	100%	100%	100%

Table 3.3: Data on the best carbon offset ratios in our study.

Experiment Results

We used empirical traces of request rates and carbon prices to study the most profitable carbon offset ratios for green datacenters over time. Recall, in Section 3.2.1, we computed η for the eastern datacenter using a constant request rate and the default offset ratios of the western US and European datacenters. In this section, we compute η for T timestamped request rates and offset ratios. Assuming low risk investing, the output reduces to a vector of T carbon offset ratios for each datacenter, where the

Metric: Accuracy ($\sum_n \frac{I(C_n=Pred(n))}{n}$)			
Web host	*Mode*	*Reactive*	*Tail Aware*
Diurnal	97%	95%	97%
Heavy tail	66%	65%	70%
Heavy tail w/ Carbon market	79%	73%	80%

Table 3.4: Accuracy of reactive and tail-aware reactive approaches. Shown for the western US host.

t^{th} setting reflects the smallest ratio c_t that maximizes $\eta_t(c_t)$ given the request rate λ_t. Our final assessment of profit uses our model to combine results from all T time steps.

Figure 3.9 shows two normalized request rate traces taken from an HP cloud application used across the world [92]. These traces cover approx. 8 days and capture diurnal patterns in the request rate. Both traces were normalized to produce about 1.5 million requests per day (about 175 RPS). They differ in the distribution of request rates within a day. The top trace matches the distribution of all arriving requests. Its 99^{th} percentile of request rates is 1.5X larger than the 99^{th} percentile of an exponential distribution with the same mean. In other words, the top trace has a tail that is only slightly heavier than an exponential distribution. The bottom trace captures the arriving requests for 1 request type. The 99^{th} percentile of request rates in this trace is 5X larger than the 99^{th} percentile of a normal distribution with the same mean. In other words, the bottom trace has a tail that is much "heavier" than an exponential distribution. Such heavy tails are a well studied in cloud applications [29, 11].

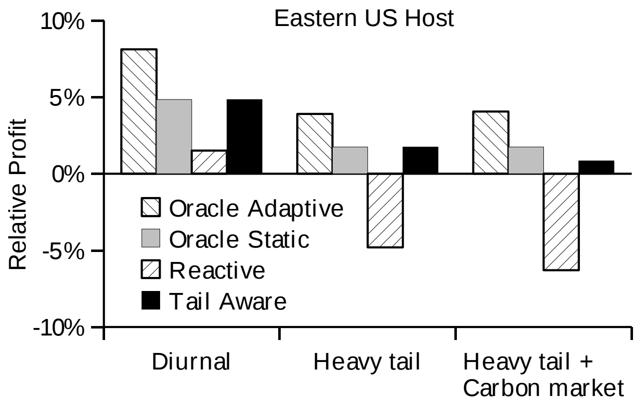
Using these traces, we compare our reactive approach to an oracle-driven adaptive approach that sets the offset ratio to the value that maximizes profit for the upcoming interval (called *oracle adaptive)*). We also compare against an oracle-driven fixed-setting approach that sets the offset ratio to the value that most frequently maximized profit throughout the trace (i.e., the statistical mode for the whole trace). These approaches use advanced knowledge that would be unavailable in a deployed system, but they are useful in demonstrating how well our reactive approach works. We also compare against the *over offsetting* approach which sets offset ratio to c_{max}. The idea behind this approach is that increasing the offset ratio will only increase η (which is not true). We make this over-offsetting approach our baseline.

3.2.3 Case Studies on Shared Hosts

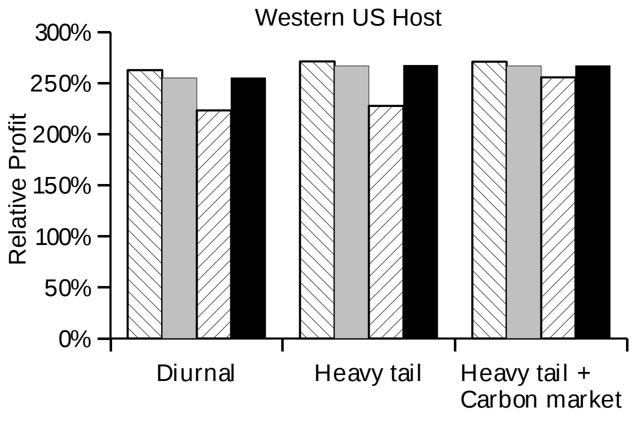
Figure 4.2 details adaptive green hosting. At every provisioning interval, the hosted application owners recently observed data on its request arrival rate to compute the offset ratios that would maximize instances leased from each host (Section 3.2.1). The adaptive green hosts keeps a history of such data, and uses it to set its offset ratio for the next interval (Section 3.2.2). The hosted application then tries to maximize throughput within a carbon budget based on each host's performance and offset ratio by balancing its workload across hosts. In Section 3.2.2, we studied the effect of adaptive offset ratio on one application, finding that green hosts can increase profit by adapting their offset ratio to the application's daily and bursty workload patterns. This section studies hosts that support many applications.

Setup

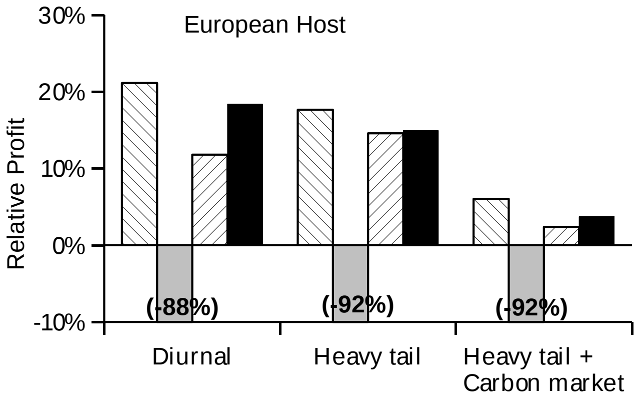
We used our VDR traces to simulate 9 Ecosia applications. Each application used a load balancer to route requests to either: 1) its best performing host, 2) the best performing host that met its carbon footprint goals, or 3) a host that offered a high



(a) Eastern US Datacenter



(b) Western US Datacenter



(c) European Datacenter

Figure 3.10: Profit of east, west, and European datacenters from the Ecosia example using real workload traces. All results are reported relative to the profit under the over-offsetting approach.

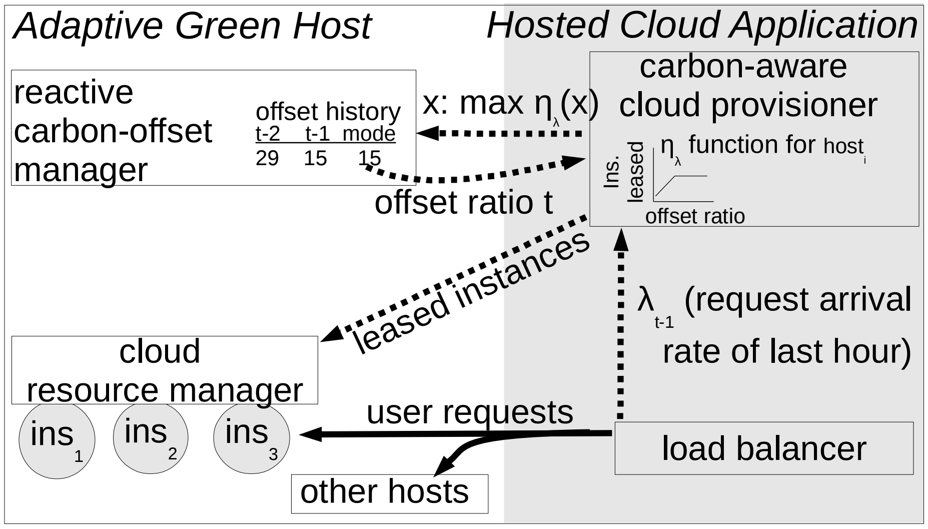
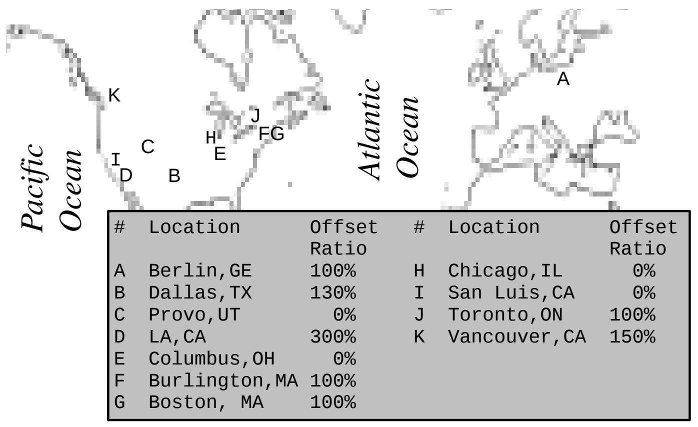


Figure 3.11: Our setup for adaptive green hosting. Dotted lines reflect data that is transmitted at every cloud provisioning interval (e.g., hourly). Solid lines reflect real time actions.

offset ratio. We defined these applications such that the best performing host mapped to one of the large Web hosts described in Figure 3.6.

Each application placed its load balancer at the best performing host and set its carbon footprint goal to the offset ratio of the fastest green host. When the load balancer sent requests to a remote host, the penalty was 1 round trip network delay (as in the queuing models in Section 3.2.1). We modeled delay between hosts using: 1) distance in miles between the other hosts and the nearest host, 2) speed of light, 3) a slowdown coefficient, and 4) TCP processing overhead. We calibrated the slowdown coefficient with regression tests on ping results between a laptop in Columbus, OH and servers deployed in London, UK, Frankfurt, GE, Berkeley, CA, St. Louis, MO, and Rochester, NY. We set the coefficient to 2.4.

Figure 3.12 plots the cities where each Web host's servers resided. The legend in the figure shows the carbon offset to dirty energy ratio offered by each host. We



#	Location	Offset Ratio	#	Location	Offset Ratio
A	Berlin,GE	100%	H	Chicago,IL	0%
B	Dallas,TX	130%	I	San Luis,CA	0%
C	Provo,UT	0%	J	Toronto,ON	100%
D	LA,CA	300%	K	Vancouver,CA	150%
E	Columbus,OH	0%			
F	Burlington,MA	100%			
G	Boston, MA	100%			

Figure 3.12: Where the shared Web hosts in our case studies live. We chose 11 of the largest Web hosts (green and traditional) using domain tools and online searches. Unintentionally, our results include hosts in North America and Europe only.

collected this data from public websites. There are 11 hosts listed, each is labeled with a letter to hide its identity. The two hosts offering the most carbon offsets (D and K) do not provide the highest throughput for any application.

Table 3.5 shows the set up for each application's load balancer and its carbon footprint goal. Two hosts (B and K) that offered offset ratios greater than 100% were used by applications with diverse carbon footprint goals. Also, one well located carbon-neutral host (J) supported diverse footprint goals. Specifically, host B supported 7 applications with the following goals: 100%, 100%, 130%, 130%, 130%, 130%, and 150%. Host J supported 3 applications with the following goals: 100%, 100%, and 130%. Finally, host K supports 6 servies with the following goals: 150%, 130%, 130%, 130%, 130%, and 130%. We used the heavy tailed VDR trace for each application (Figure 3.9). The price of carbon offsets was fixed. The maximum throughput of each node was 600 requests per second.

#	footprint goal	Best performing	Best performing + meets goals	Many Offsets
1	100%	E	J	B
2	150%	C	K	D
3	100%	H	J	B
4	130%	A	B	K
5	130%	J	B	K
6	130%	F	B	K
7	150%	I	K	D
8	130%	G	B	K
9	130%	B	K	D

Table 3.5: The configuration of each application's load balancer in our setup. The leftmost columns show the application number and its footprint goal. The rightmost columns label which hosts the application routes requests to.

At the top of every hour, our tail-aware reactive approach collected the ideal offset ratio for each application during the previous hour. We set the offset ratio for each application individually. Total profit for a host was the sum of profit from each hosted application. We compared this approach to the fixed offset policies commonly used in practice: 100%, 150%, 200%, and 300%. Here again, we call 300% the over offsetting approach and used it as our baseline.

3.2.4 Shared Hosting Results

Figure 3.13 shows the relative profit increase from our *adaptive green hosting approach*. Our approach consistently outperformed the over offsetting approach, increasing profit by at least 68% in each case. Our gains were lowest (68%) for host J because its hosted applications saw a wide difference in the offset ratio between their best performing hosts (0%) and host J (100%). Any investment in carbon offsets offered high yield. Indeed, the profit per application under the over offsetting policy

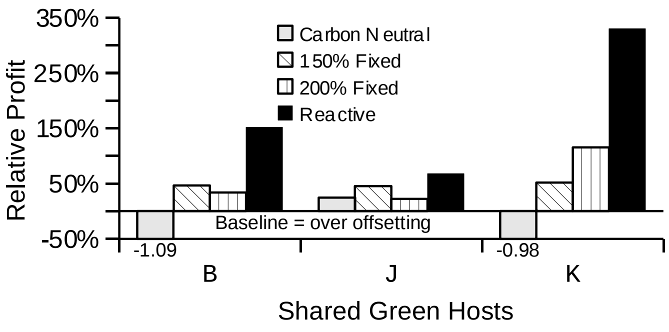


Figure 3.13: Relative profit of the shared green hosts (B, J, and K). Each host's profit per application under the over offsetting policy was \$2.17, \$7.66, and \$1.5 respectively. We used the VDR request trace with heavy tail arrival patterns (7.8 days). The over offsetting policy sets a fixed offset ratio of 300%. Recall, only hosts B, J, and K were shared by applications with diverse footprint goals.

(\$7.66) was 2–4 times larger than the other hosts. Here, our approach increases profit by adapting to workload changes in applications. We also run the same experiment on diurnal traces mentioned in Section 3.2.2. The relative profit increase for host B, J and K are 105%, 69% and 236% respectively.

We also compared two approaches commonly used in practice: over offsetting and carbon-neutral green hosting. Host B and K gain the most from over offsetting because they were in competition against other green hosts. Host J preferred a carbon neutral approach. First, our approach adapted to each host's environment, consistently outperfoming both approaches.

Is Adaptive Green Hosting Really Green? Adaptive green hosting increases profit in two ways. First, it helps green hosts buy carbon offsets with low risk, allowing them to make bold investments (up to c_{max}) to bring in customers. Second, it helps green hosts avoid wasting money on too many offsets. This latter benefit could actually make hosts less green than they are today. Figure 3.14 shows the

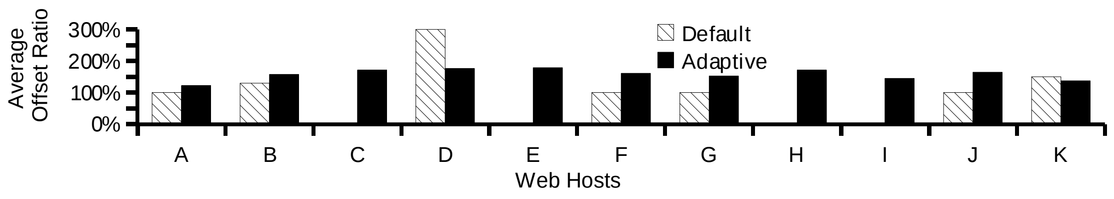


Figure 3.14: Average offset ratio recommended by adaptive green offsetting for each host in our setup.

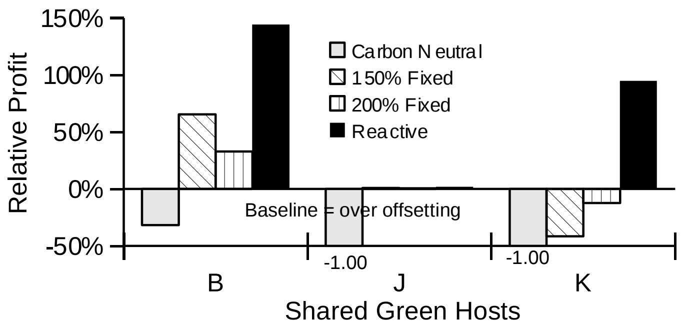


Figure 3.15: Relative profit of the shared green hosts when applications provision according to a different optimization model [107]. Each host's profit per application under the over offsetting policy was \$6.33, \$5.50, and \$4.83 respectively.

suggested average offset ratio of adaptive green hosting in our setup. The average offset ratio increased for 10 of the 11 hosts. Only host D, which offered a ratio of 300%, had a lower average offset ratio than its default. Because green hosts reflect a minority of web hosts in general, adaptive green hosting is likely to suggest increased investment in clean energy.

Can Adaptive Green Hosting be Applied to Different Service Models? The applications that we have studied so far have been based on minimizing instances (cost) within carbon and throughput constraints [66]. However, recent work has

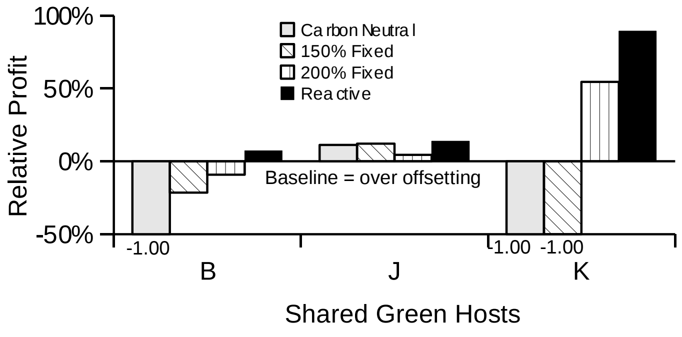


Figure 3.16: Relative profit of the shared green hosts when applications can choose to buy carbon offset directly. Each host's profit per application under the over offsetting policy was \$1.71, \$2, and \$0.66.

explored alternative models. Zhang et al. [107] proposed a model that maximizes renewable energy usage within cost and throughput constraints. Our approach to create carbon offset elasticity models can be applied to this application model also. We modified our setup to allow applications #1, 2 and 3 to use this application model. Figure 3.15 shows the results. Services in this model tend to route a few requests to the greenest datacenter. Host J (which offers on 100% offset ratio by default) suffers the most. Over offsetting helps this host the most. Hosts B and K can adapt not only to supporting diverse carbon footprint goals but even to diverse application models. Our adaptive approach increases relative profit by more than 100 percentage points for both hosts.

Is Adaptive Green Hosting Useful when Services by Offsets Directly? Instead of using green hosts, applications could buy offsets directly, removing the need to route requests across multiple datacenters. As discussed in Section 3.2.1, applications that adopt this approach lose economic benefits from bundling hosting and offsetting costs. Nonetheless, we can compute the carbon-offset elasticity for these applications by treating carbon markets as a special Web host that offers many offsets and zero throughput. We divided $cost_{co2e}$ by the price of an EC2 instance and used the result (approx. 8000%) as the offset ratio for the special, carbon-market host. We added this host as a fourth choice to every application in our setup. Some applications used this host, reducing the profit per application for the shared hosts. However, as shown in Figure 3.16, our adaptive approach still provided the most profit for shared green hosts, increasing profit by at least 7% compared to the over offsetting approach.

3.2.5 Discussion

Green hosts invest in clean energy while keeping their prices low and competitive. These hosts profit from their investment by hosting more cloud applications than their traditional counterparts; it is possible that they can tap into a niche market to accomplish this. Today's green hosts adopt ad-hoc policies for investing in clean energy, e.g., by buying as much clean energy as possible within a fixed budget. We showed that such fixed policies yield below optimal profit when the hosted cloud applications support diurnal and bursty workloads and when the hosted applications have diverse carbon footprint goals. We proposed a new research agenda: *adaptive green hosting*, where hosts invest in clean energy based on prior or predicted yield. We proposed a first-cut reactive solution that exploits heavier-than-exponential tails in cloud application workloads. Our reactive approach improves profit for existing green hosts and tends to urge hosts to increase their investments in clean energy.

Since datacenter owners could offset their carbon footprint through renewable energy credits, adaptive green hosting does not require any hardware changes in the datacenter. However, datacenter owners have to monitor and analyze their customer applications' behavior. In the next chapter, we will introduce a non-intrusive way to analyze an application's energy footprint.

Chapter 4: Black-Box Analysis for Cloud Applications

In previous chapters, we discussed about how to make cloud applications and datacenters become carbon-aware. However, nowadays, most applications and datacenters are not carbon-aware since there is little incentive to motivate them to switch to renewable energy. We believe that the demand of cloud applications' end users is the best way to urge applications and datacenters to be environmental friendly. The implementation and architecture of an application affects its energy footprint a lot. Helping end users to know more details about an application's implementation level details would be the first step to let users get involved. However, due to security and commercial considerations, it is almost impossible for application owners to reveal their application's implementation details. As a third party, users have to infer based on publicly available data.

Extracting an applications' internal information from outside is not only useful to analyze an application's energy or carbon footprint, it could also be a useful tool to detect an application's performance bugs and study live production cloud applications for third parties.

4.1 Analyzing Third-Party Applications by Decomposing Response Times

We propose a method to extract an application's implementation level information using its response times. Nowadays, a typical cloud application is a distributed system consisting of several software components collaborating together to achieve different tasks. Users interact with these applications through a request-response interface. Inside a cloud application, one or more software components will cause delays for each user request by processing or queuing the request. Such delays accumulate and are observed as response times to the end users. Our work involves decomposing response times to recover the software delays within a cloud application so that we could extract implementation level information by analyzing the recovered delays individually. Based on the recovered delay distributions, we could accurately get useful information like the type of software running behind the application; the number of parallel software instances processing the request, hence the normalized energy footprint.

Our first challenge was to model software delays for applications hosting live workloads. Our approach is black box; it does not require modifying back-end software. Instead, we used independent component analysis (ICA) to decompose response time into per-component delays. ICA is a well known machine learning algorithm that extracts source signals from multiple, independent composite signals. In our context, source signals are the delays caused by software invocation during request processing. Composite signals are response times observed from independent and parallel

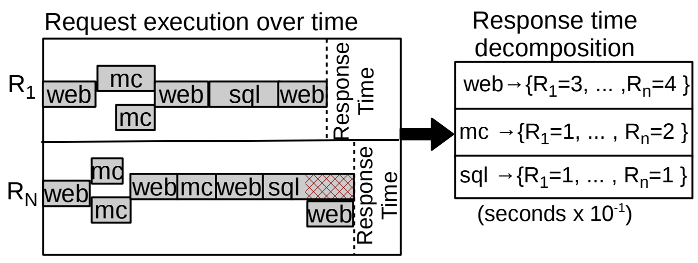


Figure 4.1: Decomposing response times into delay caused by each software component.

requests. ICA assumes component delays are non-Gaussian. This assumption is reasonable because widely used software is known to have fat tails. ICA exploits fat tails to learn about delays caused by software.

After we decomposed response times into software delays, we defined statistical confidence thresholds to prune recovered components that were unstable. Then, we labeled components using a library that included widely used software, e.g., MySQL, Redis, MongoDB, etc. Our library also models the degree of data parallelism within request executions.

4.1.1 Methodology

Figure 4.1 shows the execution of Php scripts (web), Memcached (mc), and MySQL (sql) during request processing. The web and sql components run in sequence using remote procedure semantics. The mc component uses parallel threads with each thread processing keys in its partition range. Finally, the web component triggers timeouts when the sql component is too slow. When this happens, the sql component finishes executing in the background while web completes the request.

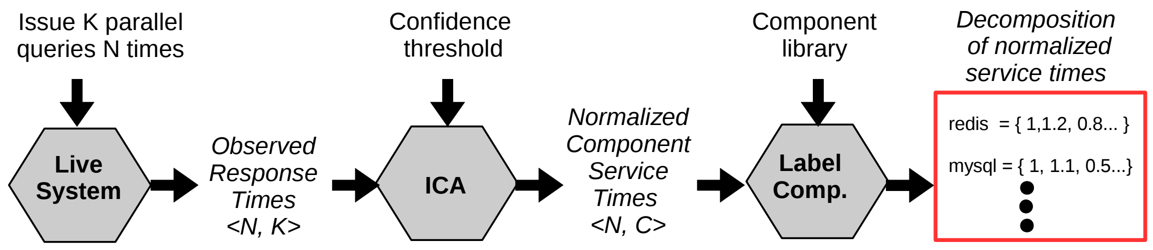

Issue K parallel queries N times
Live System
Observed Response Times <N, K>
Confidence threshold
ICA
Normalized Component Service Times <N, C>
Component library
Label Comp.
Decomposition of normalized service times
redis = { 1,1.2, 0.8... }
mysql = { 1, 1.1, 0.5...}

Figure 4.2: Our approach to decompose response time into *normalized* software delays. It does not require changing or monitoring back-end servers.

Our goal is to decompose response time into the delays caused by each software component. Figure 4.1 depicts the response time decomposition. First, we must define software delay. In our context, software components comprise a code base that is repeatedly invoked during request processing. Software delay is the portion of response time spent executing a code base *and* waiting for CPU, memory and disk resources to execute it. In Figure 4.1, the sql component does not cause software delay after web times out, because background execution does not increase response time. Likewise, the slowest mc thread causes software delay but other, parallel mc threads do not.

Prior work time stamps the start and end of software invocations to decompose response time. For this paper, we used a black box approach shown in Figure 4.2. We did not change software or operating systems on the back-end servers. First, we measure response time for a target application by issuing many requests in parallel and over time. Then, we use ICA to recover the distribution of per-component software delay. We then prune our results for stability. Finally, we use a library to label recovered components.

Limitations: Our black-box approach is useful when cloud applications block access to their back-end servers. However, it is not as powerful as direct instrumentation. By comparing Figure 4.1 and Figure 4.2, we highlight the following limitations:

- Our approach returns *normalized* software delay, not actual delays. Here, normalized means that software delays are shifted to have zero mean and unit variance. As a result, we can not directly compare delays between two recovered components.

- Our approach captures the distribution of software delay, not per-request delays. We can not explain slow response time for a specific request.

- Our approach does not recover delay for every software component used during request processing. There may be other components that affect response time as well.

Despite these limitations, our approach helps managers identify components with fat tails, label hidden components, and model energy footprints.

System Model

We model software delay as a stochastic process with a linear multiplier. Random variable s_i is software delay per invocation, where i indexes components. Components are invoked a_i times during request processing. The total software delay for a request is $a_i \times s_i$. Considering a request invokes I components, then its response time x is a random variable which is a linear combination of all invoked components' delays, i.e. $x = \sum_{i=1,\dots,I} a_i s_i$. In this paper, we use vector representation $x = \mathbf{a}^T\mathbf{s}$, where $\mathbf{a} = (a_1, \dots, a_I)^T$ and $\mathbf{s} = (s_1, \dots, s_I)^T$. Bold lowercase letters to represent vectors. Assuming per-component software delays comprise the majority of end-to-end response time, we let x_n be the response time of the n^{th} request and $\mathbf{x} = (x_1, \ldots, x_N)^T$ represents response times of N concurrent requests. Suppose the n^{th} request invokes the i^{th} component $a_{n,i}$ times, then we have:

$$\mathbf{x} = A\mathbf{s} \tag{4.1}$$

The mixing matrix A is unknown. Our goal is to find software delays ($\mathbf{s}$) by only observing the response times ($\mathbf{x}$).

Extracting Per-Component Delays with ICA

To be sure, it is impossible to solve Eq (4.1) using only response times ($\mathbf{x}$) without constraining delays ($\mathbf{s}$) or the mixing matrix A. The number of unknowns exceeds the number of observations. A key contribution for this paper is the identification of practical constraints that 1) capture common operating conditions for cloud applications and 2) allow us to solve Eq (4.1). The constraints are:

1. **Per-component delays are non-Gaussian:** It is well known that software delays in cloud applications often have fat/heavy tails [91].
2. **Normalized component delays are independent:** Software delays depend on the processing speed of their underlying hardware. However, normalizing to zero mean and unit variance, makes these delays statistically independent. Software delays also depend on queuing. However, auto-scaling [46] and multi-path networks significantly reduce queuing variance, limiting the affect of queuing delay on normalized delays.
3. **Invocation frequencies vary between requests:** Request parameters affect the invocation frequency of software like databases and memcached.

Given the above constraints, ICA can be used to recover normalized software delays using only response times. ICA reverses Eq (4.1) by finding the mixing matrix W that is most likely $W = A^{-1}$. Specifically, ICA explores candidate W matrices and chooses the matrix that minimizes mutual information between software components and Gaussianity within components. Minimizing mutual information maximizes independence of normalized delays [28], whereas minimizing Gaussianity constrains ICA to realistic mixing matrices. We use FastICA [15], one possible implementation of ICA that uses gradient descent methods to explore candidate W matrices. FastICA is a randomized, fixed-point, parameter-free algorithm whose convergence is cubic [57].

Choosing Request Parameters: Remember that the response times we collected are a set of vectors. These vectors are realizations of the random vector $\mathbf{x}$. Within each vector, there are response times from concurrent requests which invokes components with different frequency to make sure the rows in the mixing matrix A are linearly independent. Between those vectors, they should be the same set of requests to make sure that the mixing matrix A is the same across the experiment. Request parameters must be chosen carefully based on the presumed design of the target application.

Finding Stable Components

FastICA uses randomized, gradient descent. If it is executed twice on the same application, it may recover different component delays. This can happen for two reasons. First, the application can change in between the two executions. For example, CRON jobs that run only at night may shift software delays. Second, FastICA may converge upon a W matrix that is a local minima, introducing false-positive components.

We use two thresholds to build confidence that the recovered components reflect true software delays under normal operating conditions. The first threshold T_0 is a percentage, ensuring that recovered components are found in more than T_0 of FastICA executions. The second threshold T_1 sets the minimum similarity between components recovered across multiple executions. It is an absolute relative error. Recovered delays across two executions are from the same component if 1) their absolute relative error is less than T_1 and 2) for both components, there does not exist another recovered component with lower absolute error.

Labeling Components

Finally, our approach uses recovered delay distributions to infer the underlying code base. The key assumptions here are that 1) widely used software will have unique normalized delay distributions and 2) ICA can recover delays with sufficient accuracy to distinguish components. We use K-nearest neighbor clustering to match recovered delays to a library. Our library includes software components deployed with different levels of data parallelism. Thus, a match describes the recovered component's code base and energy footprint.

4.1.2 Validation

To validate the accuracy of our component delay recovery method, we compare delays for the recovered components to the observed software delays obtained by instrumenting the system. We setup a two-tier storage application. In each tier, there are several options of software to run. The first tier runs Memcached, Redis or ZooKeeper which are in-memory key-value storage software. The second tier runs

MySQL, PostgreSQL, MongoDB or ElasticSearch. This gives us 12 possible configurations for the system

We run our experiments on virtual machine instances from Google Cloud Compute Engine. For MySQL, PostgreSQL, Memcached, MongoDB and Redis, we run them on single "n1-standard-1" instances, which has one CPU core from 2.5GHz Intel Xeon E5 v2 (Ivy Bridge) and 3.75GB of memory. ZooKeeper runs in a 3-node cluster where each node is an "n1-standard-1" instance. ElasticSearch runs on a two-node "n1-standard-1" cluster. All experiments are conducted within "us-central1-f" region, which is located in Council Bluffs, Iowa.

The system accepts 6 types of requests, where each of them triggers a write operation in each tier with different frequency. In each experiment, we issue these 6 requests types concurrently and repeatedly send such concurrent requests every 500ms for 1000 times. This results 1000 sets of response times where each set contains response times from 6 concurrent requests with different types. For each configuration, we repeat this experiment for 100 times. We set $T_0 = 50\%$ and $T_1 = 3\%$ to prunce false components.

Accuracy of Recovered Component Delays

The first question we want to answer is that if ICA could accurately recover software delays. We know that ICA could recover statistically mutually independent component delays. According to the discussion in previous section, if software delays are independent, then ICA could in theory recover them.

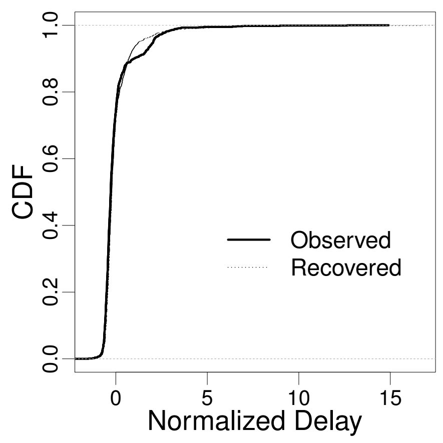
Comparing one recovered component with a set of software delay requires a metric to measure the distance of two sets of 1000 samples of delays. We use symmetric Kullback-Leibler divergence [21], or KL divergence, as our metric to measure the distance. Intuitively, given two sets of samples, KL divergence tells how many additional bits are required to represent one set of samples with another. The maximum divergence for 1000 samples is 17.667 bits. Relative error (reported as a percentage) is KL divergence divided by this number.

Given an experiment, every recovered component's delays is matched with the closest software delays monitored from the experiment. We then collect KL divergence of those matches in all experiments under all configuration. We find that the recovered component delays could be used to accurately approximate software delays. The 50th and 90th percentiles of the error between component and software delays in all experiments are 0.5% and 15.5%. Figure 4.3 shows the corresponding empirical cumulative distribution functions (eCDFs). As we discussed in Section 4.1.1, the component delays that ICA recovers are normalized to zero mean unit variance. The X-axis in these eCDFs is normalized delays. Remember that our result considers all experiments (100 experiments for each configuration) under all configurations (3 tier-1 software and 4 tier-2 software). As shown in these figures, ICA could accurately recover software delays in almost all cases.

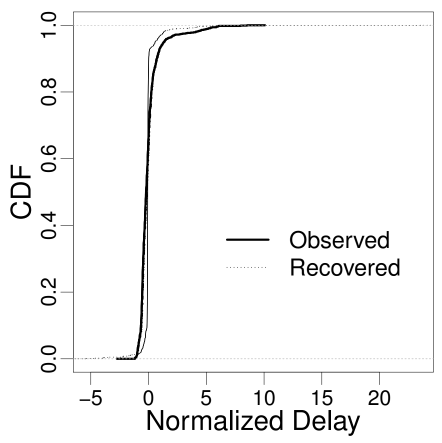
Recovered Delay Accuracy under Different Workload

In the previous subsection, our experiment is conducted under idle systems. There is no outside workload other than our probing requests. We would also want to know how well ICA recovers delays under live workloads.

We set up the same two-tier storage application again and probed the system at the same rate. This time, we issued two workloads concurrently. The first workload is fixed-rate sending 300 requests per second which contains 100 read requests and



(a) Observed and recovered delays for a component that yields median error (error=0.5%).



(b) Observed and recovered delays for a component that yields 90^{th} %tile error (i.e., near worst case,error=15.6%).

Figure 4.3: Cumulative distribution functions for software delays. Delays are normalized to zero mean unit variance.

200 write requests. The second is the WorldCup98 [10] workload with time varying workload.

Figure 4.4 shows the results of the median, 90th and 95th percentile errors of recovered delays under different workloads. We can see that adding workload in the background does not hurt the accuracy. In fact, it even improves the accuracy in the worst cases. Remember that ICA recovers component delays that are statistically mutually independent and non-Gaussian. Applying workloads skews the component delay distributions in each tier, making them less like a Gaussian distribution. Note, this test compares recovered component delays to observed delays under the same workload. Later, we will study the effect of changing the workload, i.e., comparing recovered delays under one workload to observed delays under a different workload.

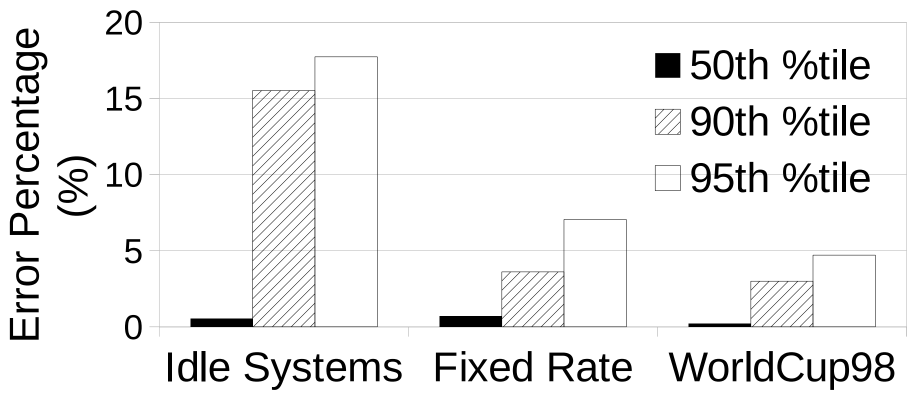


Figure 4.4: 50th, 90th and 95th percentile errors of the recovered component delays under different workloads.

Recovered Delay Accuracy for Different Software

The next question we want to answer is that if the accuracy of recovered delays differs from software by software. The answer turns out to be positive. Some software components are recovered with lower error. Figure 4.5 shows 50th, 90th and 95th percentiles of KL divergence between software and component delays grouped by software. Note that ElasticSearch, PostgreSQL and MongoDB's delays could always be recovered accurately; while Memcached and Redis' recovered component delays are bit far from their software delays. It is not surprised because fast in-memory key-value stores like Memcached and Redis have relatively small delays making it easily to be masked by other components' delays.

Remember that we issue 6 concurrent requests every time to the system making ICA could recover at most 6 components. The number 6 is chosen quite arbitrary and we would also want to see what happen if we only consider the first 2, 3, 4 or

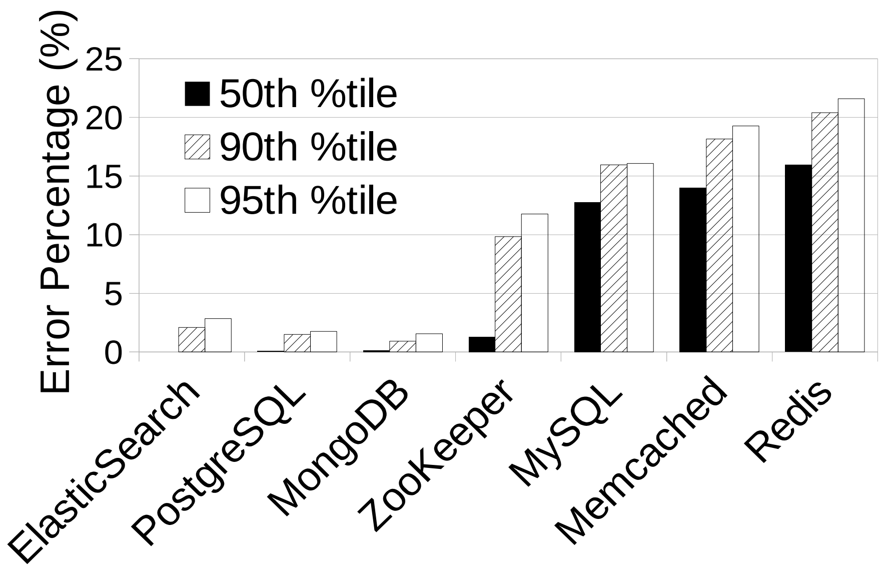


Figure 4.5: 50th, 90th and 95th percentile errors for different software. There is no background workload running on the test systems.

5 concurrent requests and only recover 2, 3, 4 or 5 components from the selected response times.

Figure 4.6 shows the 90th percentile of KL divergence for ZooKeeper and Redis under the settings of recovering 2, 3, 4, 5 and 6 components fprom the response times. First, ZooKeeper's delays are constantly better recovered than Redis' delays. This validates our statement from another angle that some software are more friendly to ICA than others. Secondly, by extracting more components, the accuracy could be slightly improved. This is partly because by extracting more components, some noise could be isolated in separate components making other components less noisy. These two trends could also be found in other software under our test.

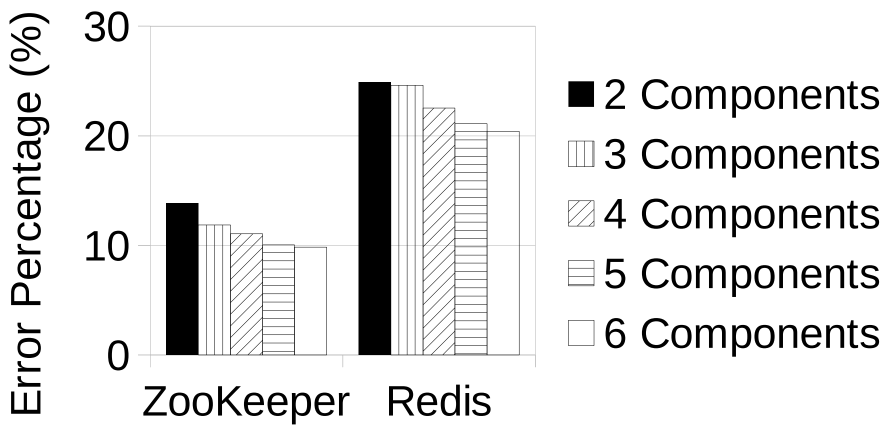


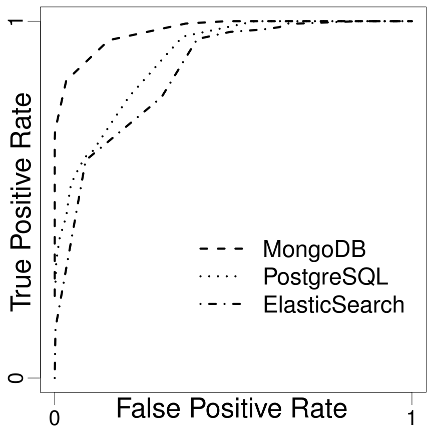
Figure 4.6: 90th percentile KL divergence for ZooKeeper and Redis by extracting 2, 3, 4, 5, and 6 components from response times.

Identifying Software by Recovered Component Delays

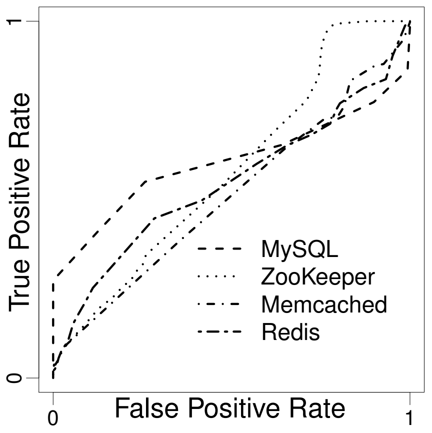
To push our study even further, we want to see if we could use recovered component delays to identify the underlying software running inside a application. That is, can we extract components accurately enough to distinguish their normalized distribution from other software.

We built a library of normalized software delays measured under idle workload for Memcached, Redis, ZooKeeper, MySQL, PostgreSQL, MongoDB and ElasticSearch. We collected 10 sets of $10,000$ delays for each software component.

To identify the underlying software in the system, we build one binary classifier for each software. Each classifier takes a recovered component delays as input and returns positive if the component delay is close enough to the corresponding software delays. The algorithm of our classifier is quite simple and could be summarized into one sentence: *Return positive if in the library there are more than K sets of software delay*



(a) ROC curves of software with low errors



(b) ROC curves of software with high errors

Figure 4.7: Receiver operating characteristic (ROC) curves of classifiers for each software component when K=3.

samples whose KL divergence with the input component delay is less than $KL_{threshold}$ *bits.* Two parameters are K and $KL_{threshold}$. Since in our library, each software has 10 sets of delays, K is between 1 and 10. As we mentioned before, the recovered component delays could not be equally accurate for all software, $KL_{threshold}$ differs from software to software. Because we build one classifier per software, there are 7 classifiers in total in our test.

To measure the performance of the classifiers, we use receiver operating characteristic (ROC) curves to show the results. ROC curve is a widely used graphical tool to illustrate the performance of a binary classifier. The basic idea of the ROC curve is to run the classifier under different parameters on the same data set and record its false positive rate and true positive rate. The Y-axis of an ROC space is the true positive rate, which is the division of number of true positives (in our case, it means the system contains the software and the classifier could correctly identifies it) over the number of positives (in our case, it means the system contains the software) in the data set. The X-axis is the false positive rate, which is the division of the number of false positives (in our case, it means the system does not have the software but the classifier falsely identified the software) over the number of positives. Both true and false positive rates are between zero and one. To draw an ROC curve, we pick a fixed value for K, change $KL_{threshold}$ to get different values of true-positive and false-positive rate and draw those points in the ROC curve.

Figure 4.7 shows ROC curves when $K = 3$. The diagonal in the ROC space means a classifier that performs random guess. Any point above the diagonal in the ROC curve means a possible configuration which is better than a random guess. We can clearly see that the performance of classifiers is highly correlated with the accuracy of the recovered component delays. We group the ROC curves in two figures: Figuer 4.7(a) shows the software whose delays could be recovered with low error. These software can be accurately identified by their classifiers. In the case of MongoDB, the true positive rate could reach 98% while the false positive rate could be still under 20%. On the other hand, software, whose recovered delays has medium error, like MySQL and ZooKeeper could be identified but sacrifices true positive rate for lower false positive rate.

However, software delay distributions will be skewed by its workload. This restricted our approach because the workload of software is unknown when the library is built. Fortunately, modern cloud computing principles mitigates this problem. Principles, especially like auto scaling, make per server workload stable by dynamically adding or removing resources to each tier when the outside workload changes.

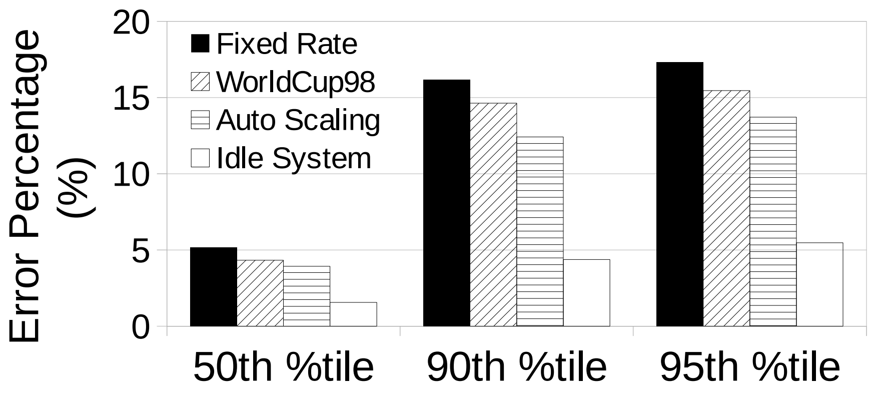


Figure 4.8: Comparing recovered components under different workload against a library. The library is built using software delays in an idle system.

We conduct an experiment on the storage system running Memcached and MongoDB in each tier serving WorldCup98 day 70's workload. Each tier scales individually along with the changes of the workload. We then compare the recovered component delays against a library built with Memcached and MongoDB delays in an idle system. As a comparison, we also run fixed-rate (300 rps) and Worldcup98 workload on another system without auto-scaling. Figure 4.8 shows the median, 90th and 95th percentile error by comparing recovered component delays with software delays in our library. We can see that auto-scaling decreases the difference.

Exploring Parallelism within Components

As we described in Section 4.1.1, in our model, a request is processed sequentially through independent components. There is no parallelism taken into consideration when we apply ICA on response times.

Since we have shown above that with a help of a library of known software, we could match the recovered components' delay back to software within the library. We would like to further use the library to explore the parallelism within a component.

Looking into one tier in the system, consider a request is being processed in parallel within a cluster of servers. Assuming that the request can only be processed by the next tier only if N servers respond the request. We call the number N as the *parallel factor* for the software in the tier. The delay of the tier is the maximum delay of the N servers, which would be recovered by ICA.

To detect the parallel factor of a given software, we adopt the same idea of using a library of known software. We could setup the software on one server under a controlled environment and collects its delays. We only keep the maximum delay of every N delays and build a new set of delays. If a recovered component's delay is close to the set of delays, we would say that the recovered component has a parallel factor of N.

To validate our approach, we setup the two-tier storage application, and put a 2-node MongoDB cluster in the second tier. For the first tier, we use one node to run Memcached. We re-use the same MongoDB delays from the training set used in Section 4.1.2 but uniformly re-sample them (with replacement) by keeping the maximum of every 2, 3, and 4 delays and construct 10 sets of delays for each parallel factor. Again, in each set, there are 1000 samples of delay.

We then apply ICA and recover independent component delays from the system. Each recovered component delay is compared with the re-sampled MongoDB delays within the training sets using symmetric KL divergence. Figure 4.9 shows the 50th and 90th percentiles of KL divergence between the recovered component delays and

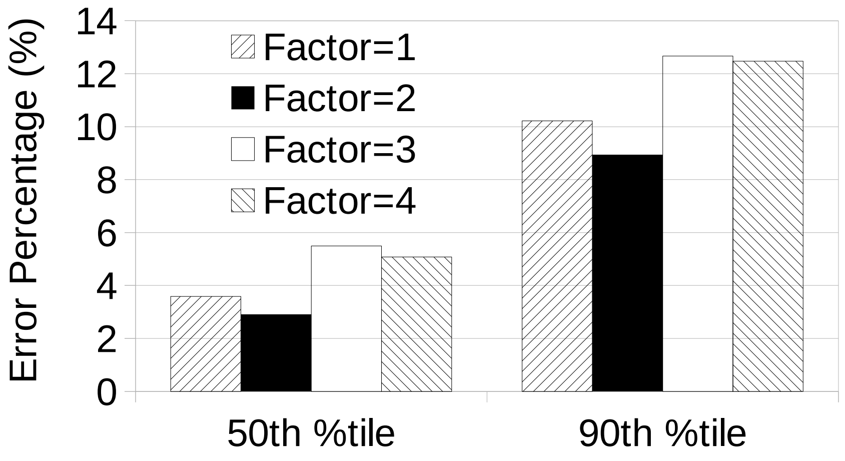


Figure 4.9: 50th and 90th percentiles of KL divergence between component delays and re-sampled MongoDB delays with parallel factors of 1, 2, 3 and 4. The actual parallel factor in the system is 2.

re-sampled MongoDB delays with parallel factors of 1 (i.e. the original MongoDB delays), 2, 3, and 4. As we can see, the re-sampled delays with parallel factor of 2 is the closest to the recovered component delays. This agrees the fact that we use a 2-node MongoDB cluster in the backend of the system.

4.1.3 Study on Real Cloud Applications

We applied our approach to decompose response time for 33 real cloud applications that support keyword search. Requests that execute keyword search meet the requirements outlined in Section 4.1.1:

1. **Keyword search uses software with non-Gaussian service times:** ElasticSearch, Apache Solr, Memcached, Redis and SQL databases are commonly used to process keyword searches. As shown in Section 4.1.2, these components have fat tail, non-Gaussian execution times.

2. **Keyword searches can have independent normalized delay:** Our search parameters include long words composed from random letters. These words subvert caches that would make normalized delay between requests inter-dependent. Each of our applications are likely to proceed through all layered caches before returning a result.

3. **Invocation frequencies vary:** Our requests also use search parameters that specify real keywords and categories. Under live workloads, these parameters ensure concurrent requests will invoke cache components differently.

We selected a wide range of applications from large popular sites like Google and Amazon to smaller sites like Sundial (a comedy magazine). All applications support HTTP/HTTPS, allowing us to use CGI to specify keyword parameters. For each application, we issued 5 concurrent requests 1000 times, allowing 500ms idle time between each round. We conducted 20 experiments for each applications, spreading experiments over 30 days. We set T_0 to 50%, meaning we pruned components that appeared in fewer than 10 experiments. We set T_1 to 3% error, meaning components were considered to represent the same software component if the absolute relative error between their distributions was less than 3%.

Component Delays and Response Times

Figure 4.10 compares tail response time and component delay for each application. Slow components do not necessarily cause slow response time. Craiglist, Youtube, Yelp, Google and Amazon use components with relatively fat tails but achieve skinny tails for response time. The 11 applications that achieve fastest response-time tails

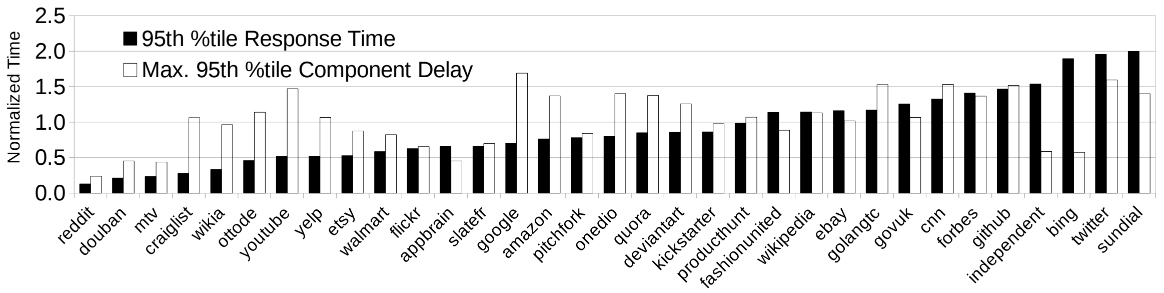


Figure 4.10: 95^{th} percentile of normalized response times compared to the largest 95^{th} percentile of recovered software delays for 33 real cloud applications.

support at least one component with a longer tail. To be sure, Figure 4.10 reports normalized delay. We can not directly compare tails for response time and components. However, it is likely that these applications are engineered to survive slow responding components. Either slow components make up a small portion of response time or the applications react when components take too long (e.g., timeouts).

Figure 4.10 also shows that applications with poor tail response time are affected by multiple components. The 5 of the 6 applications with slowest tail response time perform worse than their slowest component. Response time tail results from multiple components executing slowly at the same time.

Labeling Software Used in Real Sites

We have shown in the previous section that it is possible to use recovered component delays and a library of known software to roughly find what software is running in the back-end system given a cloud application's response times. Besides, we could further manipulate the library to find how many instances of a given software is used in parallel to process a user request. Number of parallel instances involved in each

	Found & Confirmed	Found & Unconfirmed
ElasticSearch	Ebay, Etsy, Kickstarter, Walmart, Yelp	Bing, Deviantart, Slate.fr
MongoDB	FashionUnited, Gov.uk, Mtv.com, Otto.de, Slate.fr	Ebay
PostgreSQL	AppBrain	Yelp, Walmart
MySQL	Ebay, Flickr, Github, KickStarter, Pitchfork, Twitter , Walmart, Wikia	CNN.com , Etsy, The Independent
ZooKeeper	Reddit	Etsy, Flickr, KickStarter

Table 4.1: Results of finding software running in cloud applications. *Found & Confirmed* means the software is found by our classifier and we can find at least one reliable source confirming that the application uses the software.

request could be used as a rough estimation of the energy footprint per request of the cloud application.

We apply the same technique on real cloud applications' data and try to find what software runs behind those applications. We also consider different parallel factors for each software. For each software and a given parallel factor, we build a binary classifier comparing the recovered delay with the software delays in the library. The classifiers are tuned using our two-tier storage application's data. Their parameter K is set to maximize the area under their ROC curve (AUC); and $KL_{threshold}$ is chosen to maximize the true positive rate while maintains the false positive rate less than 1/3 of the true positive rate.

Table 4.1 shows the software that we found for each cloud application. For recovered component that matched software components in our library, we searched for public-domain data that confirmed that the application actually uses the matched software component. Specifically, we searched technical blogs, official Powerpoint slides, white papers and reliable third party articles. Even though different applications are running different versions of software under diverse environments, our classifiers could still have a decent performance partly because we use normalized software delays. In most cases, once our classifier finds a software running behind a application, it could be confirmed by at least one source. However, there are some obvious errors that we can see from the table: Microsoft Bing is very unlikely to use ElasticSearch (though, they may use similar proprietary software). Yelp and Walmart use MySQL, making it unlikely that they also use PostgreSQL.

We matched 28 recovered, stable components against the open source components in our library. We were able to confirm 20 of the components were used in production by the applications (71% success rate).

Studying Normalized Energy Footprint of Real Sites

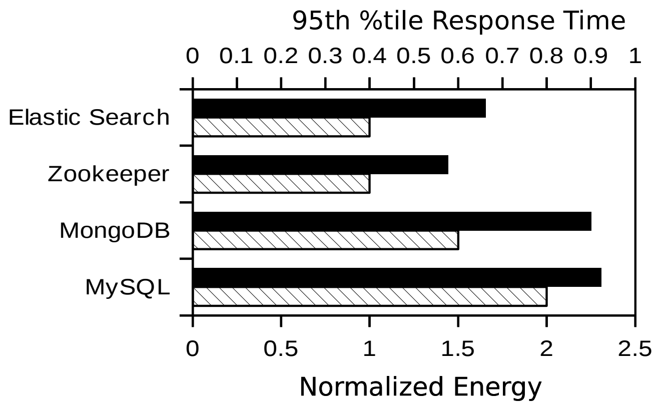


Figure 4.11: 95th percentile response times and normalized energy footprint. The black bars are the median of the 95th percentile response time of the sites that are found using the software by our classifier. The shaded bars are the median normalized energy footprints (or parallel factors) of those sites.

Recall, our matching approach discovers the parallel factor for each component in library, i.e., an estimate on the number of data-parallel software invocations during request execution. Assuming that data-parallel optimizations linearly increase energy footprint, the parallel factor describes normalized energy foorprint.

In Figure 4.11, we noticed that increasing data parallelism would increased the tail response time. We grouped energy footprints and 95th percentile response time by the software component matched in our library, reporting the median for both energy footprint and response time. As we can see in the figure, higher tail response time is highly correlated to higher energy footprints. The correlation coefficient of them is 0.908.

Chapter 5: Related Work

Our work provides solutions for different participants in the carbon-aware computing market. In literature, there are a large amount of works focusing on each of these issues.

5.1 Energy Accounting

Accurately monitoring and fairly attributing energy consumption is the first step to energy efficient systems. Bellosa [18] used a linear model to estimate CPU and memory energy consumption using data collected from hardware counters. Similar approach was used in [19] to model power consumption for multicore processors. PowerTracer [74] maps the low-level measurements back to request context. These low-level traces were combined to produce diverse views of the system ranging from per-node system call counts to per-tier energy efficiency. Power container [89] accounts and controls the energy usage of individual requests in multicore systems. It replies on a model to estimate per-core energy consumption and tracking systems for user requests. To control the energy usage, power container introduced a fair power capping that only penalizes power-hungry requests. HaPPy [106] considered hyperthread processors and modeled the energy consumption for each job in a system.

5.2 Energy Efficient Hardware

Works have been done to improve the energy efficiency at the hardware level. Weiser et al. [103] studied several algorithms to adjust CPU frequencies under different workloads. It used the CPU utilization as the major metric to control the CPU speed. This early work laid down a foundation for DVFS. Luiz et al. [17] suggested that energy proportional computer systems are desirable for datacenters, because real production servers spend most of their time at utilization between 10 to 50 percent. Their work showed that some CPUs were already energy proportional, while other components like memory, disk are not, making servers less energy efficient in most of their time. Delaluz et al. [31] investigated several hardware and software approach to control memory's power mode. They used a combination of techniques managed to save up to 89 percent of energy under some workloads. MemScale [35] introduced low-power mode for main memory by lowering the bandwidth for energy savings. It focused on multi-core systems which have multiple memory controllers. Using a heuristic algorithm, MemScale calculates the frequency combination across memory controllers to minimize overall system energy under user-specified performance constraints. Similarly, CoScale [34] considered both memory and CPU cores to heuristically find a frequency setting to save energy.

5.3 Carbon-Aware Applications

One general approach to reduce carbon footprints is to distribute workload among several datacenters. Datacenters with lower carbon footprints will be preferred if other conditions are same. Looking at the carbon footprint of entire datacenters, these works measure the contribution of renewable energy from utility providers over time. By dynamically migrating workload to the datacenter with the least carbon-heavy, grid energy, the aggregate footprint for a collection of datacenters is reduced. Le et al. [66] studied applications that capped their carbon footprints either by cap-and-trade, cap-and-pay, or absolutely capped policies. Their key insight was that a central load balancer could route requests between green and dirty Web hosts to maintain a low carbon footprint while meeting SLAs.

Liu et al. [72] provided a model to assess a datacenter's performance to carbon footprint efficiency. They use weighted linear models to find the best datacenter, proposing a scalable algorithm to do so. Zhang et al. [107] studied cloud applications that tried to minimize the carbon footprint of certain requests within a fixed budget. This approach reflects a common practice where large companies outsource a small portion of their operation to a green host, often for conspicuous altruism [54, 1].

Lin et al. [71] studied the general geographical load balancing problem by using online algorithms, which assuming only history and near future data is available and try to calculate a solution whose difference between the optimal solution (the solution considering both history and all future data) is bounded. In the paper, three online algorithms were discussed to solve geographical load balancing problem: Receding Horizon Control (RHC), Fixed Horizon Control (FHC) and Averaging Fixed Horizon Control (AFHC). The paper discussed these three algorithms, mainly RHC and AFHC, by comparing their competitive ratios under homogeneous and heterogeneous configurations. It proved that RHC provides good performance on homogeneous data centers, which is $\left(1 + \dfrac{\beta}{(w+1)e_0}\right)$-competitive, where e_0 is the cost of running an idle server, β is the switching cost of turning on a server. This means better prediction algorithm, which provides larger w, could improve the result linearly. However, on heterogeneous data centers, RHC is $\geq \left(1 + \max_s(\frac{\beta}{e_{0,s}})\right)$-competitive. Compared with RHC, AFHC gives a more robust competitive ratio, which is $\left(1 + \max_s \frac{\beta_s}{(w+1)e_{0,s}}\right)$-competitive in both homogeneous and heterogeneous settings. This means AFHC could give better worste-case results if the prediction algorithm is improved to predict further future even under heterogeneous setting.

Instead of redirecting user requests, Yank [90] migrates virtual machines to deal with the intermittent nature of renewable energy. The key idea is to reduce cost by using unreliable but cheap power for most servers, called *transient servers*; while using reliable but expensive power for small group of servers, called *stable servers*, as back ups. When the power is going to be cut, an *advanced warning* will be sent to affected transient servers. Stable servers will be used to record the state of virtual machines (VMs) running on those transient servers. The transient servers will be shut down after a short *warning period*. Yank explored a solution working for different lengths of warning period.

There are also works focusing on reducing carbon footprint for specific type of application. Blink [86] proposed a key-value storage application that transferred popular keys away from nodes that were turned off during intermittent clean energy outages. The challenge was to serve as many read and write requests as possible using only resources powered by clean energy. BlinkFS [88] used the similar idea on designing a distributed file system for intermittent power supply. GreenHadoop [51] schedules MapReduce jobs to increase the use of renewable energy. It used a model to predict the on-site renewable energy production in the near future and postpones jobs until the renewable energy is available or the jobs' deadline.

5.4 Carbon-Aware Datacenters

Ren et al. [84] modeled the energy capacity planning problem for datacenters using a linear programming. It considered renewable energy credits and multiple power sources, including power grid, on/off-site renewable energy generators, diesel generators, energy storage devices. The goal is to minimize the total cost for datacenters by smartly planning the energy usage from different sources. Li et al. [68] turned off processor cores (e.g. via DVFS) to increase the ratio of renewable energy to dirty energy on a system. Similarly, Gmach et al. [48, 49] found that server-power capping and consolidation to power servers under low renewable-energy production can enable renewable powered services, albeit with a performance cost. GreenWorks [67] is also a research of a full stack power management system in renewable data centers. It differs from previous work due to its hierarchical power management and coordination framework. The GreenWorks manages power supply in three tiers, where each tier runs a type of *green worker* specifically designed for the tier. The first, datacenter level tier runs green worker called *baseload laborer*, which controls the output of all energy sources through micro-grid central controller. The second tier is at the cluster/PDU level running *energy keeper* which monitors the battery state and control battery discharging when power supply drops or load surge happens. The third tier is the rack level running *load broker* which uses CPU frequency scaling to match the power supply. It is well known that datacenters support dynamic workloads that exhibit daily [11, 92], bursty [29], and nonstationary patterns [92]. Control theory solutions are now widely used in research and practice. Abdelzaher et al. [9] provides a good primer on such techniques, covering resource and admission controllers, sensors, and reactive and predictive techniques. Stewart et al. [94] was among the first to explore these problems, showing that datacenters must use costly batteries or grid ties to make up for below-threshold renewable-energy production.

There are experiments on building real environmental friendly datacenters. Net-Zero energy data center [13] is an experimental research project from HP lab aiming to build a data center with zero emission within its whole life time, starting from mining and producing the materials used in the data center. This requires life-cycle assessment (LCA) for all hardware including servers, buildings, cooling system, etc. To reduce the carbon emission, all components in the data center should have a low embedded exergy consumption. Because most components have positive embedded exergy consumption, this requires the data center produce more energy than its consumption in its lifecycle to offset the embedded carbon emission in their hardware. Parasol [50] is a container-size data center in Rutgers powered by solar panel and grid. It did not consider the embedded energy consumption of the data center. This gives Parasol more flexibility when building the data center. Both approaches have their own contribution. Net-zero energy data center is the most environmental friendly way of building and operating a data center. Parasol proposed a cost-centric way of using renewable energy making it more practical for current real-world data centers.

For-profit companies provide green hosting service to attract carbon aware customers. AISO [1], HostGator [56], Green Geeks, and GreenQloud [53] reflect a growing cadre of green hosts that hope to profit from their investments in clean energy. AISO, the eldest of these green hosts, was founded in 1997 but its customer base began to grow rapidly in 2002, increasing by 60% through 2008 [101]. AISO's growth marks the start of an ongoing boom in green hosting. HostGator [56], a green host based in windy Texas and founded in 2002, is now one of the largest low-cost Web hosts in the world, hosting over 1.8 million domain names. While AISO buys solar panels to invest in clean energy, HostGator buys renewable energy credits from local wind farms. The latter approach, using renewable energy credits. Datacenter owners could invest in renewable energy farms or buy credit from renewable energy market; while the renewable power is merged into the electrical grid. The benefit of this approach is to use renewable energy without initial installation. It also allows hosts like HostGator to support offset ratios greater than 100% by buying multiple credits for every joule used. HostGator in particular offsets 130% of the dirty energy used to power its servers. Green Geeks offset 300%. Several sources also suggest that Google is currently buying renewable energy credits to offset its datacenters' carbon footprints [1, 38].

5.5 Cloud Application Performance Analysis

Chapter 4 introduced a black-box approach to analyze a cloud application's per-component performance. This approach helps end users to get implementation level details about an application. There are several methods available to study a cloud application's performance from different perspectives requiring different amount of exposure of the application. To the best of our knowledge, our ICA-based approach is the first work that analyzes a cloud application from per-component's perspective but only requires information available to third party users.

Works have been done to trace user requests within a cloud application's back-end system. Frameworks like Pinpoint [26] monitor components triggered by each user requests and locate faulty components using data mining techniques. Tracing user requests to see which components are triggered by a request is also required in systems like Magpie [14], X-Trace [44] and EntomoModel [95]. In these systems, each component is monitored by either changing their source code, or their running environment, e.g. using a modified kernel. Low level information like system call trace, CPU utilization and memory usage is also used to study live cloud applications' behavior. PerfScope [30] analyzes recent system calls to perform online bug inference. It narrows down the possible buggy functions by detecting time or frequency changes in system calls. PREPARE [96] monitors virtual machines' system-level metrics and applies statistical learning algorithms to detect performance anomalies. It works at the hypervisor level making it possible to be applied by cloud providers. Li et al. [69] studied the relationship between low-level metrics, like system call trace, CPU utilization and memory usage, and the tail latencies of components in cloud applications. Their findings shown that scheduling policy, CPU power saving mechanisms and NUMA effects would significantly affects the tail latencies. Software log messages are also useful to conduct a component-level study of a cloud application. Tools like DISTALYZER [77] helps users to discover anomalies by comparing log from normal runs against logs with anomaly performance. Using source code information along with log messages would even help users to locate the exact line of anomaly. Both Xu et al. [104] and Ghanbari et al. [47] use static analysis to find log statements in the source code and relate the anomalous log messages back to the source code. When source code is not available, other approaches such as Mantis [64] and ConfAid [12] modified application binaries to collect events. All these approaches are useful for cloud applications' administrators who has access to the back-end system.

Chapter 6: Conclusion

Carbon emissions increasingly affect the design and management of cloud applications and datacenters. Our research has shown that carbon-aware management, at every layer in the cloud computing stack, can increase profit and improve end-user satisfaction.

We proposed a general policy and mechanism for carbon-aware applications and introduced carbon-awareness into storage applications. Our experiments showed that carbon-aware applications prefer heterogeneous systems, where they choose resources from several datacenters whose resources combined could reduce carbon footprint. Because different types of applications may exhibit different characteristics, there may be opportunities to further reduce their carbon footprints by fully exploiting their potentials in a case-by-case study. Instead of studying general frameworks for carbon-aware applications, the future work for carbon-aware applications would focus more on application-specific features, so that the application could be further tuned to perform a specific task under low carbon footprints. Some examples could be low-carbon key-value in-memory data store, low-carbon video streaming service, etc.

We also studied power delivery mechanisms to allow datacenters to concentrate their carbon offset investments to to servers used by carbon-aware users. Our study first focused on an architectural level solution to concentrate renewable energy into a group of resources. We identified a key component in renewable datacenters: *grid ties*, the device most commonly used to integrate renewable energy into the datacenters power delivery system. Our research showed that placement of grid ties would dramatically affect the number of *renewable powered instances* — a metric we proposed to measure concentration of renewable energy. Second, we proposed *adaptive green hosting*, a system for datacenters which adaptively control its renewable energy supply to cloud applications. We argued that renewable energy investment should directly lead to profit. Our approach is to distribute renewable energy to carbon-aware applications to encourage them to use more renewable-powered resources.

There are still open problems for adaptive green hosting. Future work will improve upon our approach by considering more complex interplay between SLAs and carbon footprint goals, heterogeneous energy efficiency and carbon efficiency among hosts, and in depth workload prediction approaches. Besides, interactions between multiple adaptive green hosts would be an interesting topic. In this case, when a green host tries to set its offset ratio, it has to consider the offset ratios set by other competing hosts using similar adaptive approach.

Finally, we devised a novel approach to infer a cloud application's implementation and energy footprint using only publicly available data. Solving this problem may encourage cloud applications to be carbon-aware by exposing their internal information to their end users. Such information is also useful for third parties to diagnose an application's performance issue. We presented a series of study on decomposing response times into per-component delays. Using Independent Component Analysis (ICA), response times can be accurately divided into component delays with 90th percentile error less than 17%. Using a library of known software, these recovered component delays can be used to find the software running behind an application and its corresponding parallel factor, which is directly related to per-request energy footprint. Further, 33 real cloud applications' per-component delays are studied along with their response times. We found that the collaboration between components in a service is important to the system's performance. Also, we found that the energy footprint and tail response time are correlated.

Looking from different perspectives of all participants in green computing market, our research showed that, even if renewable energy is more expensive than traditional dirty energy, it is possible and profitable to invest into this field. Considering the carbon emissions of the global IT sector [61], encouraging more companies to reduce their carbon footprints would benefit the society in a large scale.

Bibliography

[1] Aiso.net: Web hosting as nature intended. http://aiso.net.

[2] Amazon elastic compute cloud. `http://aws.amazon.com/ec2/pricing/`.

[3] Apache web server. http://httpd.apache.org/.

[4] Ecosia - number of daily searches. `http://ecosia.org/statistics.php`.

[5] Green House Data: Greening the data center. `http://www.greenhousedata.com/`.

[6] ipath global carbon etn. `http://www.ipathetn.com/product/GRN/`.

[7] TPC. `http://www.tpc.org/`.

[8] Google solar panel project. `http://www.google.com/corporate/solarpanels/home`, June 2007.

[9] Tarek Abdelzaher, John Stankovic, Chenyang Lu, Ronghua Zhang, and Ying Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, 2003.

[10] Martin Arlitt and Tai Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30–37, 2000.

[11] Martin F Arlitt and Carey L Williamson. Web server workload characterization: The search for invariants. In *ACM SIGMETRICS Performance Evaluation Review*, volume 24, pages 126–137. ACM, 1996.

[12] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *OSDI*, pages 237–250, 2010.

[13] Prithviraj Banerjee, Chandrakant Patel, Cullen Bash, Amip Shah, and Martin Arlitt. Towards a net-zero data center. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 8(4):27, 2012.

[14] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In *OSDI*, volume 4, pages 18–18, 2004.

[15] Allan Kardec Barros and Andrzej Cichocki. A fixed-point algorithm for independent component analysis which uses a priori information. In *Neural Networks, 1998. Proceedings. Vth Brazilian Symposium on*, pages 39–42. IEEE, 1998.

[16] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[17] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, (12):33–37, 2007.

[18] Frank Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, pages 37–42. ACM, 2000.

[19] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 147–158. ACM, 2010.

[20] Janusz Bialek. Tracing the flow of electricity. *IEE Proceedings-Generation, Transmission and Distribution*, 143(4):313–320, 1996.

[21] Sylvain Boltz, Eric Debreuve, and Michel Barlaud. knn-based high-dimensional kullback-leibler distance for tracking. In *Image Analysis for Multimedia Interactive Services, 2007. WIAMIS'07. Eighth International Workshop on*, pages 16–16. IEEE, 2007.

[22] Ward Bower, Chuck Whitaker, W Erdman, M Behnke, and M Fitzgerald. Performance test protocol for evaluating inverters used in grid-connected photovoltaic systems. *California Energy Commission*, 2004.

[23] Christopher P Cameron, William E Boyson, and Daniel M Riley. Comparison of pv system performance-model predictions with measured pv system performance. In *Photovoltaic Specialists Conference, 2008. PVSC'08. 33rd IEEE*, pages 1–6. IEEE, 2008.

[24] Stephen L Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in SCILAB.* Springer, 2006.

[25] John Chattaway. Rackspace green survey. In *Rackspace Hosting White Paper*, June 2008.

[26] Mike Y Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 595–604. IEEE, 2002.

[27] Gary Cook and Jodie Van Horn. How dirty is your data? a look at the energy choices that power cloud computing. *Greenpeace (April 2011)*, 2011.

[28] Thomas M Cover and Joy A Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[29] Mark E Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *ACM SIGMETRICS Performance Evaluation Review*, 24(1):160–169, 1996.

[30] Daniel J Dean, Hiep Nguyen, Xiaohui Gu, Hui Zhang, Junghwan Rhee, Nipun Arora, and Geoff Jiang. Perfscope: Practical online server performance bug inference in production cloud computing infrastructures. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–13. ACM, 2014.

[31] Victor Delaluz, Mahmut Kandemir, Narayanan Vijaykrishnan, Anand Sivasubramaniam, and Mary Jane Irwin. Hardware and software techniques for controlling dram power modes. *Computers, IEEE Transactions on*, 50(11):1154–1173, 2001.

[32] Nan Deng, Christopher Stewart, Daniel Gmach, and Martin Arlitt. Policy and mechanism for carbon-aware cloud applications. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 590–594. IEEE, 2012.

[33] Nan Deng, Christopher Stewart, and Jing Li. Concentrating renewable energy in grid-tied datacenters. In *Sustainable Systems and Technology (ISSST), 2011 IEEE International Symposium on*, pages 1–6. IEEE, 2011.

[34] Qingyuan Deng, David Meisner, Arup Bhattacharjee, Thomas F Wenisch, and Ricardo Bianchini. Coscale: Coordinating cpu and memory system dvfs in server systems. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pages 143–154. IEEE, 2012.

[35] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F Wenisch, and Ricardo Bianchini. Memscale: active low-power modes for main memory. *ACM SIGARCH Computer Architecture News*, 39(1):225–238, 2011.

[36] Domain Tools. http://www.domaintools.com/.

[37] Sandhya Dwarkadas, J. Robert Jump, and James B. Sinclair. Execution-driven simulation of multiprocessors: Address and timing analysis. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 4(4):314–338, 1994.

[38] EcobusinessLinks. Green webhosts. `http://www.ecobusinesslinks.com/green_webhosts/`.

[39] Ecosia - the green search. `http://www.ecosia.org/`.

[40] Enphase Energy. Enphase grid tie inverters. http://www.enphase.com/.

[41] Environmental Leader. Data centers power up savings with renewable energy. `http://www.environmentalleader.com/2009/07/29/data-centers-power-up-savings-with-renewable-energy/`.

[42] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[43] Christopher B Field, Vicente R Barros, MD Mastrandrea, Katharine J Mach, MA-K Abdrabo, N Adger, YA Anokhin, OA Anisimov, DJ Arent, J Barnett, et al. Summary for policymakers. *Climate change 2014: impacts, adaptation, and vulnerability. Part a: global and sectoral aspects. Contribution of working group II to the fifth assessment report of the intergovernmental panel on climate change*, pages 1–32, 2014.

[44] Rodrigo Fonseca, George Porter, Randy H Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, pages 20–20. USENIX Association, 2007.

[45] Anshul Gandhi, Yuan Chen, Daniel Gmach, Martin Arlitt, and Manish Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8. IEEE, 2011.

[46] Anshul Gandhi, Sherwin Doroudi, Mor Harchol-Balter, and Alan Scheller-Wolf. Exact analysis of the m/m/k/setup class of markov chains via recursive renewal reward. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, pages 153–166. ACM, 2013.

[47] Saeed Ghanbari, Ali B Hashemi, and Cristiana Amza. Stage-aware anomaly detection through tracking log points. In *Proceedings of the 15th International Middleware Conference*, pages 253–264. ACM, 2014.

[48] Daniel Gmach, Yuan Chen, Amip Shah, Jerry Rolia, Cullen Bash, Tom Christian, and Ratnesh Sharma. Profiling sustainability of data centers. In *Sustainable Systems and Technology (ISSST), 2010 IEEE International Symposium on*, pages 1–6. IEEE, 2010.

[49] Daniel Gmach, Jerry Rolia, Cullen Bash, Yuan Chen, Tom Christian, Amip Shah, Ratnesh Sharma, and Zhikui Wang. Capacity planning and power management to exploit sustainable energy. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 96–103. IEEE, 2010.

[50] Íñigo Goiri, William Katsak, Kien Le, Thu D Nguyen, and Ricardo Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 51–64. ACM, 2013.

[51] Íñigo Goiri, Kien Le, Thu D Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 57–70. ACM, 2012.

[52] GoodEnergy. `http://www.goodenergy.com`, 2012.

[53] Greenqloud — the worlds first truly green compute cloud. http://www.greenqloud.com.

[54] Vladas Griskevicius, Joshua M Tybur, and Bram Van den Bergh. Going green to be seen: status, reputation, and conspicuous conservation. *Journal of personality and social psychology*, 98(3):392, 2010.

[55] Nielsen Holdings. The nielsen global online environmental survey, 2011.

[56] Hostgator: Web hosting services. http://hostgator.com.

[57] Aapo Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *Neural Networks, IEEE Transactions on*, 10(3):626–634, 1999.

[58] Oscar H Ibarra and Chul E Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 22(4):463–468, 1975.

[59] Raj Jain. *The art of computer systems performance analysis*. John Wiley & Sons, 2008.

[60] John Mattson, Emerson Inc. Personal communication. `http://emerson-datacenter.com`.

[61] James M Kaplan, William Forrest, and Noah Kindler. Revolutionizing data center energy efficiency. Technical report, Technical report, McKinsey & Company, 2008.

[62] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[63] Jonathan G Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):034008, 2008.

[64] Yongin Kwon, Sangmin Lee, Hayoon Yi, Donghyun Kwon, Seungjun Yang, Byung-Gon Chun, Ling Huang, Petros Maniatis, Mayur Naik, and Yunheung Paek. Mantis: Automatic performance prediction for smartphone applications. In *Proceedings of the 2013 USENIX conference on Annual Technical Conference*, pages 297–308. USENIX Association, 2013.

[65] Larry O'Connor, CEO of OWC.net. Personal communication. `http://owc.net`.

[66] Kien Le, Ricardo Bianchini, Thu D Nguyen, Ozlem Bilgir, and Margaret Martonosi. Capping the brown energy consumption of internet services at low cost. In *Green Computing Conference, 2010 International*, pages 3–14. IEEE, 2010.

[67] Chao Li, Rui Wang, Tao Li, Depei Qian, and Jingling Yuan. Managing green datacenters powered by hybrid renewable energy systems. In *International Conference on Autonomic Computing (ICAC)*, 2014.

[68] Chao Li, Wangyuan Zhang, Chang-Burm Cho, and Tao Li. Solarcore: Solar energy driven multi-core architecture power management. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 205–216. IEEE, 2011.

[69] Jialin Li, Naveen Kr Sharma, Dan RK Ports, and Steven D Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014.

[70] Kevin Lim, Parthasarathy Ranganathan, Jichuan Chang, Chandrakant Patel, Trevor Mudge, and Steven Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pages 315–326. IEEE, 2008.

[71] Minghong Lin, Zhenhua Liu, Adam Wierman, and Lachlan LH Andrew. Online algorithms for geographical load balancing. In *Green Computing Conference (IGCC), 2012 International*, pages 1–10. IEEE, 2012.

[72] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H Low, and Lachlan LH Andrew. Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 233–244. ACM, 2011.

[73] LPSolve. `http://www.lpsolve.com`.

[74] Gang Lu, Jianfeng Zhan, Haining Wang, Lin Yuan, and Chuliang Weng. Powertracer: Tracing requests in multi-tier services to diagnose energy inefficiency. In *Proceedings of the 9th international conference on Autonomic computing*, pages 97–102. ACM, 2012.

[75] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[76] David Meisner and Thomas F Wenisch. Stochastic queuing simulation for data center workloads. In *Exascale Evaluation and Research Techniques Workshop*. Citeseer, 2010.

[77] Karthik Nagaraj, Charles Killian, and Jennifer Neville. Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 26–26. USENIX Association, 2012.

[78] National Renewable Energy Laboratory. NREL: Western wind resources dataset. `http://wind.nrel.gov/Web_nrel/`, 2009.

[79] M. Ontkush. Plethora of options for green web hosting. `www.treehugger.com`, 2007.

[80] Clayton R Paul. *Fundamentals of Circuit Analysis*. Wiley, 2000.

[81] Cameron Potter, Debra Lew, Jim McCaa, Sam Cheng, Scott Eichelberger, and Eric Grimit. Creating the dataset for the western wind and solar integration study (usa). *Wind Engineering*, 32(4):325–338, 2008.

[82] PricewaterhouseCoopers Inc. Going green:sustainable growth strategies. `www.pwc.com/en_GX/gx/technology/pdf/going-green.pdf`, 2011.

[83] Parthasarathy Ranganathan and Phil Leech. Simulating complex enterprise workloads using utilization traces. In *10th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*. Citeseer, 2007.

[84] Chuangang Ren, Di Wang, Bhuvan Urgaonkar, and Anand Sivasubramaniam. Carbon-aware energy capacity planning for datacenters. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 391–400. IEEE, 2012.

[85] SAP. Sap insight: Product environmental footprint and ecolabeling. August 2009.

[86] Navin Sharma, Sean Barker, David Irwin, and Prashant Shenoy. Blink: managing server clusters on intermittent power. In *ACM SIGPLAN Notices*, volume 46, pages 185–198. ACM, 2011.

[87] Navin Sharma, Jeremy Gummeson, David Irwin, and Prashant Shenoy. Cloudy computing: Leveraging weather forecasts in energy harvesting sensor systems. In *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, pages 1–9. IEEE, 2010.

[88] Neelam Sharma, David Irwin, and Prashant Shenoy. A distributed file system for intermittent power. In *Green Computing Conference (IGCC), 2013 International*, pages 1–10. IEEE, 2013.

[89] Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, Xiao Zhang, and Zhuan Chen. Power containers: An os facility for fine-grained power and energy management on multicore servers. In *ACM SIGPLAN Notices*, volume 48, pages 65–76. ACM, 2013.

[90] Rahul Singh, David E Irwin, Prashant J Shenoy, and Kadangode K Ramakrishnan. Yank: Enabling green data centers to pull the plug. In *NSDI*, pages 143–155, 2013.

[91] Christopher Stewart, Aniket Chakrabarti, and Rean Griffith. Zoolander: Efficiently meeting very strict, low-latency slos. In *ICAC*, pages 265–277, 2013.

[92] Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting nonstationarity for performance prediction. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 31–44. ACM, 2007.

[93] Christopher Stewart and Jing Li. Power provisioning for diverse datacenter workloads. In *Workshop on Energy Efficient Design*, 2011.

[94] Christopher Stewart and Kai 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.

[95] Christopher Stewart, Kai Shen, Arun Iyengar, and Jian Yin. Entomomodel: Understanding and avoiding performance anomaly manifestations. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 3–13. IEEE, 2010.

[96] Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Chitra Venkatramani, and Deepak Rajan. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 285–294. IEEE, 2012.

[97] Michael J Todd. The many facets of linear programming. *Mathematical Programming*, 91(3):417–436, 2002.

[98] Trefis Team. Amazon kills it in cloud computing but it wont budge the stock price. `http://www.forbes.com`, 2011.

[99] GreenPeace USA. Facebook status update: Renewable energy now. `http://www.greenpeace.org/usa/news/facebook-update-renewable-ene`.

[100] GreenPeace USA. Victory! facebook 'friends' renewable energy. `http://www.greenpeace.org/international/en/news/features/Victory-Facebook-friends-renewable-energy/`.

[101] The AMD Opteron Processor Helps AISO. www.vmware.com.

[102] Bryan Walsh. Your data is dirty: The carbon price of cloud computing. `http://time.com/46777/your-data-is-dirty-the-carbon-price-of-cloud-computing/`, 2014.

[103] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Mobile Computing*, pages 449–471. Springer, 1996.

[104] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132. ACM, 2009.

[105] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Dynamic energy estimation of query plans in database systems. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 83–92. IEEE, 2013.

[106] Yan Zhai, Xiao Zhang, Stephane Eranian, Lingjia Tang, and Jason Mars. Happy: Hyperthread-aware power profiling dynamically. In *USENIX ATC*, volume 14, pages 211–217, 2014.

[107] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware 2011*, pages 143–164. Springer, 2011.