Carbon-Aware Temporal Data Transfer Scheduling Across Cloud Datacenters

Elvis Rodrigues, Jacob Goldverg, Tevfik Kosar
Department of Computer Science and Engineering
University at Buffalo (SUNY), Amherst, NY 14260, USA
Email: {elvisdav, jacobgol, tkosar}@buffalo.edu

***Abstract*—Inter-datacenter communication is a significant part of cloud operations and produces a substantial amount of carbon emissions for cloud data centers, where the environmental impact has already been a pressing issue. In this paper, we present a novel carbon-aware temporal data transfer scheduling framework, called LinTS, which promises to significantly reduce the carbon emission of data transfers between cloud data centers. LinTS produces a competitive transfer schedule and makes scaling decisions, outperforming common heuristic algorithms. LinTS can lower carbon emissions during inter-datacenter transfers by up to 66% compared to the worst case and up to 15% compared to other solutions while preserving all deadline constraints.**

***Index Terms*—Carbon-aware scheduling, temporal shifting, data transfers, cloud datacenters.**

I. Introduction

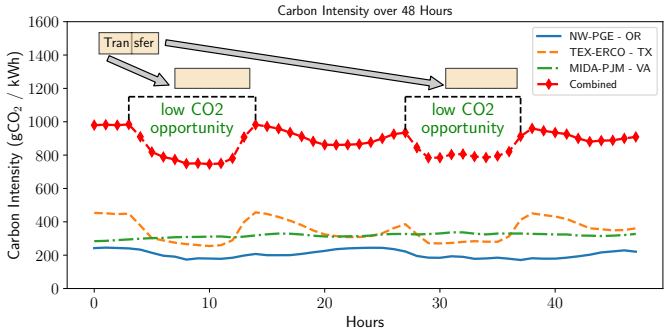
Demand for power in the U.S. is projected to grow by 2.4% over the next 7 years, of which datacenters are a significant component with a 15% projected growth till 2030 [1]. Much of this demand today stems from AI training and inference with large AI models, emerging new cloud services and network activity, and diminishing gains of power efficiency with new iterations of datacenter hardware. In response, technology companies and cloud providers have set ambitious targets to achieve net-zero, and even net-negative, carbon emissions by 2030 [2–4]. With increasing demand from AI workloads, significant investments have also been made in emerging green power generation methods such as Small Modular Reactors (SMRs) [5]. However, despite these investments, a large portion of the energy consumed by cloud datacenters is generated from brown, carbon-intense sources due to the lack of reliable renewable energy supply, undermining decarbonization efforts worldwide [2, 6]. Alongside efforts to make datacenter hardware more power efficient, there have been many studies into the nature and trends of datacenter workloads and strategies to make these workloads more carbon efficient [7–11].

Among all the workloads executed by a cloud provider, a significant amount of time and resources are spent transferring datasets between different regions and managing these datasets to maximize availability for clients and users while minimizing costs to providers. Global inter-datacenter traffic has exceeded 1.4 Zettabytes and is projected to grow by 30% every year [12]. This surge in global inter-datacenter traffic has also made the energy consumption and carbon footprint of data transfers a critical concern for cloud datacenters, where the environmental impact has already been a pressing issue. The datacenters are projected to consume over 500 TWh of energy in 2025, emitting roughly 225 metric megatons of CO_2 calculated from the global average carbon intensity of 450 gCO_2 per kWh [13], and serious efforts are needed to curb the emissions of inter-datacenter communication. Although networking technologies have advanced significantly in recent years, data transfers over networks remain highly energy-intensive and contribute substantially to carbon emissions. A study reported that sending hard drives between remote institutions via airplanes is much less carbon-emitting than transferring the data over communication networks [14].

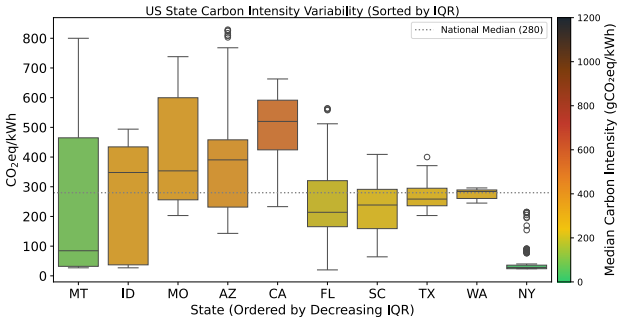
Given the urgency of reducing carbon emissions of cloud datacenters, cloud providers must now also consider minimizing the carbon footprint of the data transfer tasks. There are several strategies at the disposal of cloud providers and application developers to achieve this, starting with the sustainable design of the datacenter itself with factors such as the embodied carbon cost and failure rate of hardware, energy suppliers, and cooling solutions to consider [8]. Cyclical variations and regional differences in the sources of the power grid's energy supply can be studied to exploit temporal and spatial opportunities to schedule data transfers and place dataset replicas to further reduce carbon emissions. For this purpose, tools like $ElectricityMaps$ [15] and $WattTime$ [16] have become widely popular. Inter-datacenter transfers are often time-flexible and interruptible and thus can benefit from careful scheduling and preemption [17]. For instance, a study shows that 91% of all inter-datacenter traffic at Baidu's datacenters is replication-related and delay-tolerant, which is considered to corroborate the traffic pattern of other large-scale cloud service providers [18].

While there is existing work focusing on the placement of computing tasks and cloud workloads at low-intensity regions and time zones [9, 19–23], most works do not consider the non-negligible carbon emissions during data transfers across datacenters. This paper presents a novel approach to construct the carbon-aware scheduling of data transfer tasks as a Linear Programming (LP) optimization problem and uses standard LP solvers that are efficient and easy to integrate and deploy. More specifically, this paper makes the following contributions:

- It introduces a novel carbon-aware data transfer scheduler, LinTS, for inter-datacenter traffic.



(a) Portioning and scheduling of data transfers over 48 hours with source, intermediate, and destination sites.



(b) Variability of carbon intensity across regions and over time (24-hour period).

Fig. 1: Spatial and temporal variations in carbon intensity.

- It defines carbon-aware temporal scheduling of data transfers as a linear programming (LP) problem.
- LinTS can make scaling scheduling decisions for transfer requests, unlike common heuristic scheduling algorithms.
- LinTS can lower carbon emissions during inter-datacenter transfers by up to 66% compared to the worst case and up to 15% compared to other solutions.

The paper is organized as follows: Section II provides background into carbon-aware workload shifting strategies and discusses the related work in this area; Section III describes the motivation and foundation for applying LP to carbon-aware transfer scheduling; Section IV presents and discusses evaluations of our LP scheduler in simulation and real-world scenarios; and Section V concludes the paper.

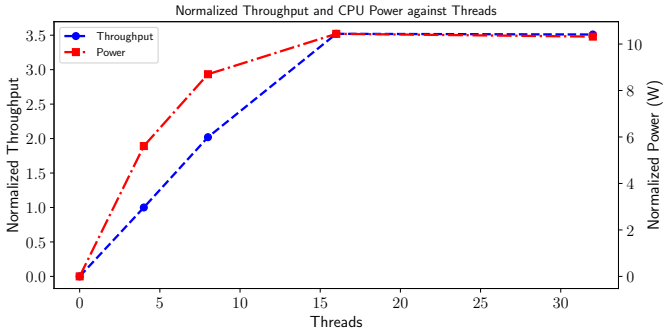
II. Background and Related Work

The spatial and temporal variances in the carbon intensity present an opportunity for minimizing the carbon emission of datacenter workloads by scheduling jobs to geographical locations and at times of the day/week with low carbon intensity through historical data and forecasts, and adjusting task parameters when grid conditions change. As seen in [7], between 8% to 31% in carbon emissions relative to the global average can be made using spatial and temporal approaches and even more when combined. These approaches, however, presume variances in regional grid carbon intensity to exploit and their viability may change as the grid's sources become greener and more uniform. The nature of the workload also impacts how suitable spatial and temporal approaches are. Long-running, data-intensive, batch workloads like machine learning (ML) training are often delay tolerant and interruptible and benefit more from temporal approaches. On the other side, bursty and interactive workloads often benefit more from spatial approaches when user responsiveness and latency are important.

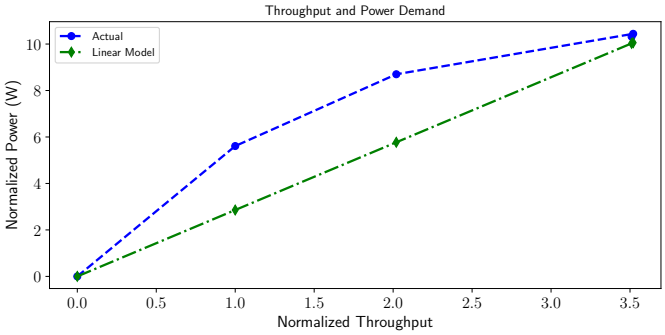
Carbon intensity for a region can vary over time due to variances in the availability of green sources provided by the grid. For instance, without battery storage, solar power can be highly available on a sunny morning but less so at night, requiring brown sources to substitute for. Similarly, carbon intensity can vary over weeks and months due to seasonal variations. Thus, recurring workloads would need to be carefully managed to minimize carbon emissions year-round. Figure 1(a) shows an example of temporal variations of the carbon intensity of an end-to-end network path, where a workload or network transfer can be scheduled in low-carbon opportunities. Some works like [24] and [20] tune the power draw of hardware with voltage and frequency scaling, by lowering voltage or frequency in periods of high intensity and increasing them in low-intensity periods. The same scaling principle can be applied to containers, Kubernetes pods, and nodes as seen in [25] and [22]. Works like [11] and [23] approach carbon optimization with capacity planning, while *WaitAwhile* [26] uses a scheduler that defers, interrupts, and resumes workloads when possible. Our work extends the ideas found in *WaitAwhile* and *CarbonScaler* [17] for carbon-aware network transfer temporal scheduling.

Among spatial strategies for carbon optimization, most commonly fall into the category of scheduling and real-time migration and often mix these two ideas in their implementations. As a workload arrives, a scheduler can use historical data or carbon intensity forecasts of various datacenters and schedule that workload in the greenest region possible, balancing factors such as deadlines, latency requirements, and the overall load. Figure 1(b) shows the variability of carbon intensity across different regions and over time (24-hour period). In some implementations, a monitoring service watches for large deviations in real-time carbon and load data while this workload is executed and may migrate the task or redirect the client to another datacenter [11, 27]. A variant of this strategy involves sharing a workload among multiple datacenters if it permits, allowing one to better balance carbon efficiency, datacenter load, and availability [28]. CADRE is a carbon-aware replication planner that makes spatial decisions and places replicas in low-intensity regions [19]. Workloads that are WAN-intensive can benefit from carbon-aware routing and overlay networks, though it is harder to measure this strategy's impact since the routers, switches, and other intermediary nodes along the path must be accounted for.

All of these strategies require a way to measure or esti-



(a) Impact of threads on CPU power demand and throughput.



(b) Correlation between the achieved throughput and power demand.

Fig. 2: Modeling the correlation between power demand and achieved throughput.

mate power consumption and a source for carbon intensity data either from power grid providers or through third-party services like $ElectricityMaps$ and $WattTime$. A workload may sometimes be more CPU-intensive, I/O-intensive, or network-intensive so some works also seek to isolate the power consumption of a node's component for better accuracy. Power measurement of the CPU package can often be done in software by reading model-specific registers (MSR) or through models with tools like $perf$ [29], RAPL [30], NVML [31], FECoM [32], and $powerletrics$ [33].

While lowering carbon emissions is the main goal for many of these strategies, it is also important to ensure that datacenter performance, availability, and reliability are not compromised. Thus, performance metrics like throughput, tail latency, scalability, datacenter load, service level agreement (SLA) violations, and cost of operations are often measured to show any effects and tradeoffs of these green strategies [34]. Likewise, due to the varying nature of datacenter workloads, some green strategies target workloads with high delay tolerance while others focus on low tolerance workloads.

Most of the carbon-aware works discussed here study datacenter workload scheduling and management and use strategies that are transferable to network data transfer scheduling, but there are a few important distinctions between the problems. Unless the compute task is geographically distributed, most datacenter tasks discussed in these works are executed in one location and only have to account for the carbon intensity of that region, while data transfers will often span multiple hops across different regions, requiring consideration of the carbon intensities of multiple zones at once. Another important distinction is the lack of control over resources in network transfers. Datacenters have control over all the resources needed to execute the task and can tune those resources appropriately, while users often do not enjoy control over routing decisions and intermediate nodes of a data transfer. Additionally, the data transfers can be affected by background data traffic activity, making potential capacity planning and throughput predictions very challenging.

Several works on datacenter workload management can be found that employ spatial strategies, temporal strategies, or both. Some works are concerned primarily with scheduling of tasks and use static optimization techniques with power models, carbon and demand forecasts, or power models to produce execution plans with sometimes mechanisms to reevaluate plans when forecast errors exceed a threshold [9, 11, 26, 28, 35]. Other works instead focus on real-time optimization, using admission control strategies [24], hardware voltage and frequency scaling [20, 24, 25, 36], horizontal scaling [17, 23], reinforcement learning agents [27], and migration [37–41].

Like LinTS, works like [42], [20], [21], and [43] use some form of linear, quadratic, and constraint programming in their schedulers with great degrees of success. However, none of these works consider the carbon cost of data transfers between cloud datacenters. While the works discussed above focus on datacenter workloads without consideration of inter-datacenter communication, as seen in replication and distributed storage applications, for example, LinTS explores carbon optimization for workloads with significant network transfers between datacenters using the carbon intensity of the network path itself.

III. LinTS Overview

We introduce the data transfer temporal scheduling problem here, where a scheduler must assign time slots and set threads, if possible, for a set of transfer requests with file sizes $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and deadlines $\mathcal{D} = \{D_1, \ldots, D_n\}$ while minimizing the total carbon emissions of these transfers. This entails accounting for all nodes along the transfer path. To make carbon-aware decisions, the scheduler can use historical carbon intensity traces or forecasts.

Notation	Description
$\rho_{i,j}$	Throughput of request i at time slot j
$\vec{\rho}$	Throughput of all requests at all slots as a flattened vector
L	First-hop bandwidth limit of the path
s_ρ	Throughput scale constant
s_P	Power scale constant
$c_{i,j}$	Combined carbon intensity of request i at time slot j
$\mathbf{c}_i$	Combined carbon intensity vector of request i
J_i	Size of transfer request i in bytes
D_i	Deadline of request i in number of slots from origin
Δ_τ	Length of time slot in seconds

TABLE I: The list of notations used in the paper.

A. LinTS Linear Programming Model

Linear Programming (LP) is an optimization technique used to minimize or maximize a linear objective function constrained to a set of linear inequalities. This can be expressed in the standard form where one solves for $\mathbf{x}$:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \preceq \mathbf{b}, \\ \text{and} \quad & \mathbf{x} \succeq \mathbf{0}, \end{aligned}$$

where A is a matrix that expresses the linear inequality and $\mathbf{c}$ is the cost vector to optimize for [44]. Thus, to apply linear programming, a linear cost function is needed. Works that study the relationship between network traffic and energy consumption commonly use either a non-linear model, a linear model, or a state-based model where power consumption increases in steps at certain throughput levels [45]; for carbon-aware temporal transfer scheduling, we adopt the linear model in this work. For this work, we assume that all requests received by the scheduler are delay-tolerant, interruptible, and schedulable.

Our solution, the **Lin**early-optimized **T**ransfer **S**cheduler (LinTS), relies on the assumption that network throughput through wide-area networks (WANs) and CPU power demand have a linear-like relationship when throughput is less than the bottlenecked capacity. To compare the linear model to the observed relationship, transfers are run on Chameleon Cloud [46] from TACC in Texas to Chicago. We scale the number of threads (and sockets) exponentially from 4 threads to 32 threads and measure the achieved throughput and corresponding power consumption. As seen in Figures 2(a) and 2(b), while throughput and power draw have a non-linear relationship with the number of threads and with each other, a linear model can be used to approximate the relationship between throughput and power demand while the network path is not saturated or congested.

If L is the bandwidth limit of the path and s_ρ the throughput scale, we model the throughput ρ achieved with θ threads with the following equation:

$$\rho(\theta) = L\left(1 - \frac{1}{s_\rho L\theta + 1}\right) \tag{1}$$

Similarly, given the maximum power P_{max}, minimum power P_{min}, and power scale s_P, we model the CPU power P drawn with θ threads as follows:

$$\Delta_P = P_{\text{max}} - P_{\text{min}} \tag{2}$$

$$P(\theta) = \Delta_P\left(1 - \frac{1}{s_P\Delta_P\theta + 1}\right) + P_{\text{min}} \tag{3}$$

While linear programming is useful for choosing times to start, interrupt, and resume transfers, the power and throughput models in Equations 1 and 3 can be used to extend the scheduler to make thread scaling decisions for additional savings in carbon intensity similar to [17]. Linear program solutions cannot be restricted to integers, so the scheduler cannot directly place threads in time slots. Instead, LinTS makes decisions on the required throughput for each slot and then uses the inverse of Equation 1 to convert throughput to threads:

$$\theta(\rho) = \frac{1}{Ls_P}\left(\frac{\rho}{L-\rho}\right) \tag{4}$$

Substituting Equation 4 into Equation 3, we get the following relation between power draw and throughput:

$$K = \frac{s_P\Delta_P}{s_\rho L} \tag{5}$$

$$P(\rho) = P_{\text{max}} + \frac{\Delta_P(\rho - L)}{(K-1)\rho + L} \tag{6}$$

where K is a constant. Restricting throughput to $0 \le \rho \le L$, we can linearize the equation:

$$P(\rho) = \frac{\Delta_P}{L}\rho + P_{\text{min}} \tag{7}$$

Expressing Equation 7 in vector terms where a vector element corresponds to a time slot, we get the following linear objective function:

$$\mathbf{c}_i^T\mathbf{p} = \frac{\Delta_P}{L}\mathbf{c}_i^T\vec{\rho} + \mathbf{p}_{\min} \tag{8}$$

where $\mathbf{c}_i$ is the carbon intensity vector for job i. Equation 8 implies that carbon emissions can be minimized by minimizing throughput, but it does not capture the fact that slower transfers take longer to complete and can often increase overall carbon emissions; this can be accounted for with linear constraints.

B. LinTS Constraints

The following constraints are required to produce a feasible plan for a given set of transfer requests. Let $\rho_{i,j}$ be the throughput of a request i at time slot j, also written as a flattened throughput vector $\vec{\rho}$ in the following constraints.

Deadline constraint. Each request has a deadline D without slack. Although this cannot be expressed as an inequality that an LP solver can use, we can encode deadline constraints through the dimensions of the throughput vector.

$$\dim\vec{\rho} = \sum_i D_i$$

Time-slot constraint. This constraint ensures that the LP solvers allocate enough throughput and time slots to complete the transfer request of size J. If a time slot is of length Δ_τ, then for each request i and time slot j,

$$J_i \le \sum_{j=1}^{D_i} t_{i,j}\cdot\rho_{i,j} \quad \forall 1 \le i \le n$$

$$\text{where} \quad t_{i,j} = \begin{cases} 0 & \text{if slot } j \text{ not part of request } i \\ \Delta_\tau & \text{if slot } j \text{ part of request } i \end{cases}$$

Thread-limit constraint. This constraint ensures that the sum of all request bandwidth allocated at a time slot does not exceed the bandwidth limit L. Then, for each slot j,

$$\sum_{i=1}^{n} \delta_{i,j} \cdot \rho_{i,j} \leq L \quad \forall 1 \leq j \leq \max_i(D_i)$$

$$\text{where} \quad \delta_{i,j} = \begin{cases} 0 & \text{if request } i \text{ not part of slot } j \\ 1 & \text{if request } i \text{ part of slot } j \end{cases}$$

Input constraint. Since LinTS assumes a bandwidth limit due to bottlenecks, the throughput constraint $0 \leq \rho_{i,j} \leq L$ ensures that the throughput cannot exceed this limit.

Taken together, the temporal scheduling problem can be expressed as the following linear program.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{D_i} c_{i,j} \cdot \rho_{i,j} \\ \text{subject to} \quad & J_i \leq \sum_{j=1}^{D_i} t_{i,j} \cdot \rho_{i,j} \\ \text{and} \quad & \sum_{i=1}^{n} \delta_{i,j} \cdot \rho_{i,j} \leq L \\ \text{and} \quad & \dim \vec{\rho} = \sum_{i=1}^{n} D_i \\ \text{and} \quad & 0 \leq \rho_{i,j} \leq L \\ & \forall 1 \leq i \leq n, 1 \leq j \leq \max_i(D_i) \end{aligned}$$

C. LinTS Implementation

LinTS is implemented in Python using SciPy's efficient `linprog` solver [47], and is simulated in Python using historical carbon intensity traces. It is designed to integrate with data transfer services as a Python library or a REST API with Flask.

Algorithm 1 is the core workflow of LinTS. Lines 1 to 5 prepare the cost vector from the weighted trace sums and encode the deadline constraint through dimensions. Lines 6 to 12 and 20 construct the deadline constraint, and lines 13 to 19 and 21 prepare the byte constraint. Finally, lines 22 to 24 call the ScipPy LP solver, unwrap the solution, and convert it to threads with Equation 4. Given these constraints, LinTS allows multiple transfer requests to share a time slot as long as their collective throughput does not exceed the bandwidth limit and is free to scale transfer threads up or down as needed.

After producing a plan, the simulator estimates its carbon emissions with the carbon intensity trace and power curve seen in Equation 3. Noise is added to the trace to emulate possible errors in carbon forecasts before iterating over the plan. If a slot has no threads allocated for any request, then the simulator assumes no energy consumption at that time as we want to measure only energy consumed by the transfer requests.

IV. Evaluation

In this section, we evaluate our solution, LinTS, in reducing the carbon emissions of data transfers across regions, comparing it to known scheduling algorithms.

Algorithm 1 LinTS Algorithm

1: forecast_sums $\leftarrow \sum$ weights $\cdot$ forecast
2: **for** f in forecast_sums **do**
3: $\quad f \leftarrow$ ExpansionMatrix $\cdot f$
4: **end for**
5: $\mathbf{c} \leftarrow$ flatten(forecast_sums)
Require: $\dim c = \sum D_i$
6: $A_{ub} \leftarrow$ [] {Upper bound linear constraints}
7: offset $\leftarrow 0$
8: **for** n from 0 to num_jobs **do**
9: $\quad$ byte_sum_vec $\leftarrow \mathbf{0}$
10: $\quad$ byte_sum_vec[i] $\leftarrow$ slot_time $\quad \forall \text{offset} \leq i \leq D_n$
11: $\quad A_{ub} \leftarrow \text{append}(A_{ub}, \text{byte\_sum\_vec})$
12: **end for**
13: **for** i from 0 to max(deadlines) **do**
14: $\quad$ slot_constraint $\leftarrow \mathbf{0}$
15: $\quad$ **for** all slots S at time i **do**
16: $\quad\quad$ slot_constraint[S] $\leftarrow 1$
17: $\quad$ **end for**
18: $\quad A_{ub} \leftarrow \text{append}(A_{ub}, \text{slot\_constraint})$
19: **end for**
20: $b_{ub} \leftarrow -8 \cdot$ data_size_vec
21: $b_{ub} \leftarrow \text{append}(b_{ub}, \text{target\_thrpt\_vec})$
22: thrpt_plan $\leftarrow \text{linprog}(\mathbf{c}, A_{ub}, b_{ub}, (0, L))$
23: thrpt_plan $\leftarrow$ unflatten(thrpt_plan)
24: thread_plan $\leftarrow \theta$(thrpt_plan)
25: **return** thread_plan

A. Experimental Setup

Carbon intensity traces. We use 72-hour slices of historical carbon intensity data from $ElectricityMaps$ consisting of hourly measurements for every power zone in the US for all of 2024 [15]. We pick sites with the highest variability in carbon intensity, namely zones in New Mexico, Colorado, Utah, Wyoming, South Dakota, South Carolina, and Montana as seen in Figure 1(b). While $ElectricityMaps$ provides carbon intensity forecasts that would commonly be used in a scheduler, it is currently limited to 0 – 72 hours depending on the subscription plan, making it infeasible for large data transfers spanning over several days or with generous deadlines. Accounting for errors with forecasts, we add random noise of 5% and 15% to our historical traces. LinTS in its current form does not adjust plans as network and carbon conditions change, leaving this capability for future work.

Node and data transfer characteristics. We simulate a data transfer over up to 8 nodes: a source, up to six intermediate nodes (i.e., router, switch, repeater, etc.), and a destination. While multiple routers and switches are often found in the data transfer path, we choose a simple WAN of up to 8 nodes connected by a long wired network to simplify the simulation. The bandwidth of the link between the source and destination is known and fixed to 1 Gbps, which we call the 'first-hop bandwidth', but the capacity of other links is unknown. Since

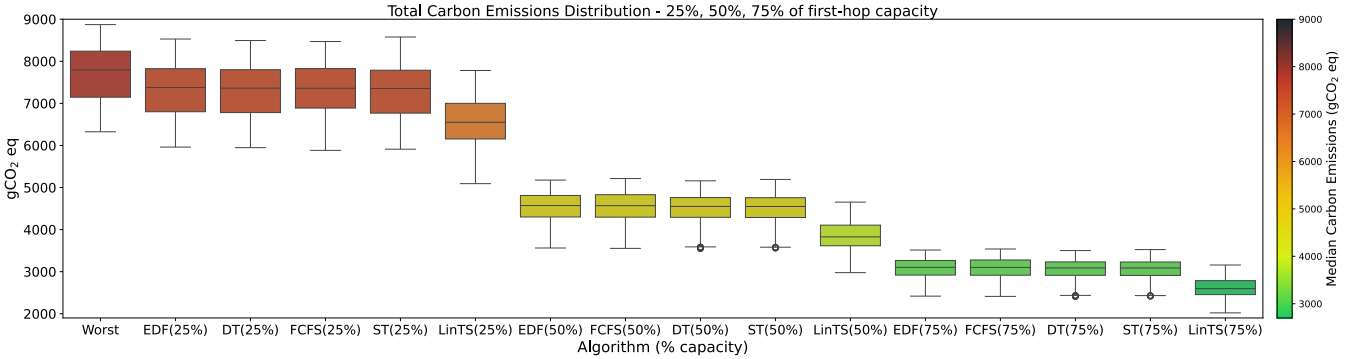


Fig. 3: Comparison of algorithms' carbon emissions when restricted to 25%, 50%, and 75% of first-hop capacity.

one can expect bottlenecks in the data path over a WAN, the plans produced by the algorithms assume a bottleneck expressed as a percentage of the first-hop bandwidth. Network throughput is assumed to scale non-linearly with threads as seen in Figure 2(a). Similarly, the power consumption of the node is assumed to scale non-linearly with threads and can range from 88W to 100W.

Algorithm configurations. Our evaluation compares LinTS to heuristic algorithms described in [48] and our best heuristic algorithm where they all produce plans for a set of transfer requests with paths and deadlines defined. These plans are evaluated in our simulator to calculate their carbon emissions. All of the heuristic algorithms below assign the highest number of threads allowed by the request's bottleneck to its time slots; this is sufficient since elapsed time is typically the dominant component of a request's footprint. Let J be the number of jobs, S the number of time slots, and L the bandwidth limit.

1) *First-come First-serve (FCFS)* – This is the default scheduling algorithm for most file transfer services. As the transfer requests arrive over a network, they are arranged in a queue in the order determined by arrival time. Then, for each request in this order, FCFS simply schedules it without optimizing for carbon footprint by assigning the first S time slots where S is the minimum number of slots needed to complete the request before its deadline; this repeats until the queue is empty.
2) *Earliest-Deadline First (EDF)* – The EDF algorithm does not optimize for carbon footprint. The transfer requests are sorted by deadline in ascending order to determine priority. Then, a request with the earliest deadline is assigned to the first S time slots where S is the minimum number of slots needed by the request. The algorithm then picks the request with the next earliest deadline and repeats this process.
3) *Worst-Case* – When the transfers are scheduled in a carbon-agnostic manner, in the worst-case scenario, the transfers can potentially be scheduled to the time slots with the highest carbon intensity. To emulate this case and establish a baseline, we use the EDF algorithm to schedule requests at time slots with the highest carbon intensities. Then, plans are generated randomly and the worst-performing plan between the two methods is used as the worst case.
4) *Single Threshold (ST)* – This algorithm uses one carbon intensity threshold to allocate time slots to a data transfer. First, the transfer requests are sorted in ascending order of deadline to determine priority. If the carbon intensity at a point in time falls below this threshold, then ST blocks that time slot and allocates it to the request. This continues until the request has sufficient time slots to complete before its deadline. Then, ST moves to the next request and repeats this process. In our implementation, the optimal threshold is found through a binary search since the plan with the lowest feasible threshold is ideal.
5) *Double Threshold (DT)* – Instead of one threshold, DT uses a high and low threshold to offset the overhead delay of resuming transfers with lower carbon intensity. Like ST, the requests are sorted by deadline. At a time slot, if the request was been paused at the previous slot and the carbon intensity is below the high threshold, then the slot is given to the request. If the request was paused in the previous slot and the carbon intensity is below the low threshold, then the slot is given to the request. We set α, the difference between the thresholds, to 50 and used binary search to find the optimal thresholds.
6) *LinTS* – The request size, deadline, and carbon intensity are written as linear constraints and fed to a solver to produce a throughput plan over 72 hours. This plan is then converted to the corresponding threads plan using Equation 4. In our evaluations, we set the bandwidth limit L to 0.25 Gbps for 25%, 0.5 Gbps for 50%, and 0.75 Gbps for 75%. All evaluations use a throughput scale s_ρ of $1/24$. SciPy's default solver switches between the simplex and interior-point methods depending on the size of the input and constraints.

Transfer requests. The algorithms above are used to schedule 200 transfer requests received and queued at roughly the same time; this point in time is the origin and we set $t = 0$. The file sizes range from 10 GB to 50 GB with deadlines ranging from 48 to 71 hours from origin.

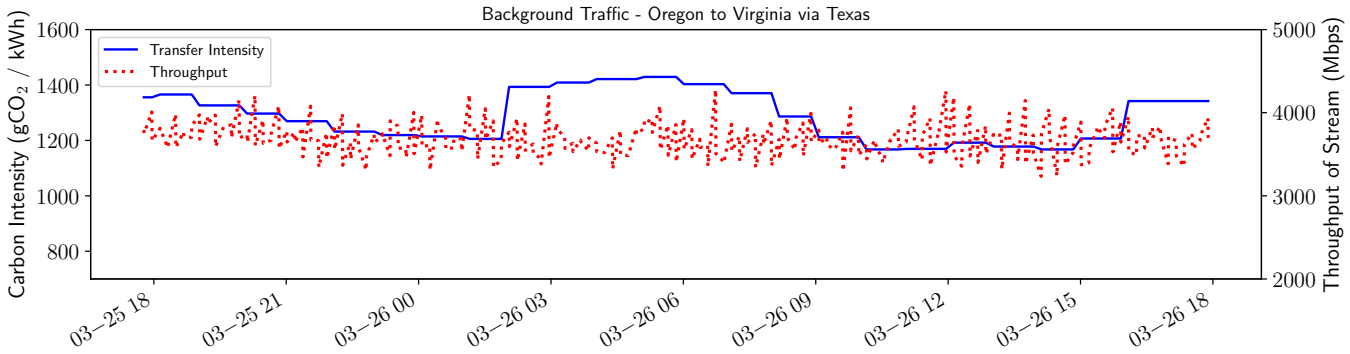


Fig. 4: Throughput and carbon intensity of transfers from AWS US-West-2 to US-East-1 via TACC over a 24-hour period.

Bandwidth Limit	Worst Case	Earliest Deadline	FCFS	DT	ST	**LinTS**
25%	7.14 kg	6.75 kg	6.76 kg	6.75 kg	6.74 kg	**6.08 kg**
50%		4.12 kg	4.11 kg	4.09 kg	4.09 kg	**3.56 kg**
75%		2.80 kg	2.79 kg	2.77 kg	2.77 kg	**2.42 kg**

TABLE II: Average carbon emissions of algorithms at 25%, 50%, and 40% of the first-hop bandwidth with 5% error. The best-performing algorithm is shown in bold.

Bandwidth Limit	Worst Case	Earliest Deadline	FCFS	DT	ST	**LinTS**
25%	7.69 kg	7.30 kg	7.30 kg	7.29 kg	7.28 kg	**6.56 kg**
50%		4.51 kg	4.52 kg	4.48 kg	4.48 kg	**3.84 kg**
75%		3.06 kg	3.07 kg	3.04 kg	3.04 kg	**2.61 kg**

TABLE III: Average carbon emissions of algorithms at 25%, 50% and 75% of the first-hop bandwidth with 15% error. The best-performing algorithm is shown in bold.

Simulator. Given a request's path of a source, intermediate node, and destination, the 72-hour carbon intensity traces for their regions are read, and divided and expanded into 288 time slots, 15 minutes each. The request path's intensity is then calculated as the sum of the traces; since we assume all nodes in the path are equally affected by network transfers, we assign equal weight to these nodes. The transfer requirements and traces are then fed into the algorithms above to produce thread plans. The CPU power demand of each plan is then estimated using Equation 3. For our evaluations, $P_{\text{max}} = 100$W and $P_{\text{min}} = 88$W. The power scale s_P is set to $1/50$. Energy consumption and carbon emissions are calculated using the traces. Noise is added to the traces to model errors in forecasts.
Evaluation metric. Since all of the evaluated algorithms produce feasible plans given transfer sizes and deadlines, we compare the total emissions and resiliency of the algorithms.

B. Evaluating LinTS

The potential and differences of these algorithms are highlighted best when there is high variability in carbon intensity over time. The worst-case emission is computed by comparing emissions of the worst heuristic and random solution searches and taking the larger of the two values. Table II shows the total carbon emissions of the algorithms when restricted to 25%, 50%, and 75% of the first-hop bandwidth (1 Gbps) with 5% noise added to the traces when evaluated, and Table III shows total carbon emissions for each algorithm but with 15% noise added instead; in both scenarios, LinTS outperforms the other algorithms with 10.1% lower carbon emissions at 25% capacity, 14.2% lower emissions at 50% capacity, and 15.4% lower emissions at 75% capacity when compared to FCFS, averaging results from the 5% and 15% error scenarios. Compared to the worst-case, LinTS achieves 14.8%, 50.1%, and 66.1% lower emissions at 25%, 50%, and 75% capacity respectively on average.

LinTS outperforms both ST and DT in all capacity settings with 9.8%, 13.6%, and 13.5% lower emissions respectively on average. Figure 3 highlights these differences with the distribution of emissions of each algorithm with 15% noise added and capacity limited to 25%, 50%, and 75% of first-hop bandwidth respectively. The solutions generated by LinTS in general save more carbon than the heuristic algorithms as indicated by the lower median and quartiles, given that LinTS allows multiple jobs to run in a time slot and that it makes scaling decisions with threads unlike the other algorithms here.

C. Impact of Scheduling on Performance

In a WAN setting, it is possible for two or more transfers to share links along their paths, which can lead to link contention and congestion. Thus, background network activity can negatively affect the capacity and bottleneck of the transfer path at any point in time. With enough demand along the path at time t, a scheduler that starts or resumes a transfer at t can potentially increase its total carbon emission since it may slow down and need more time to complete; deadline SLAs may be violated too. If congestion improves when the transfer is scheduled to start or resume, it may finish faster and yield higher carbon savings. Our evaluations above do not account for these possibilities since all of the tested scheduling algorithms do not measure or predict network congestion. While setting a conservative bottleneck capacity can help construct plans resilient to congestion, this is not necessarily a foolproof solution for highly stochastic networks.

Figure 4 shows the carbon intensity and throughput of $Iperf$ transfers [49] from AWS US-West-2 in Oregon to AWS US-East-1 in Virginia through TACC in Texas serving as the intermediate node over a 24-hour period. $Iperf$ is

run repeatedly with a fixed number of sockets to assess any significant background traffic and potential congestion between these sites, and throughput is measured every 10 minutes. Hourly carbon intensity data is collected at each hop's location, starting from Oregon, then Washington, Texas, Georgia, New York, New Jersey, and finally Virginia; these locations are determined through the traceroute utility. This intensity data is then combined into a single intensity trace of the path as a weighted sum.

In our case, the throughput varies from 3.2 to 4 Gbps over 24 hours, which can adversely affect plans produced by any scheduler. As AWS allows users to temporarily boost throughput for a price, it is possible for a surge in network activity to cause congestion and further hurt performance. Considering these changes in network characteristics and their impact on transfer scheduling, a forecast or monitoring service is needed to predict or measure background activity and reevaluate the schedule which is often seen in some datacenter workload scheduling-related work [23]. This is a potential research extension left for future work.

V. Conclusion and Future Work

We present LinTS, a scheduler that intelligently schedules and scales data transfers between cloud datacenters by constructing a linear optimization problem with carbon intensity forecasts while respecting request deadlines and requirements. Our simulations show LinTS outperforming threshold methods commonly seen in other works while remaining lightweight and easy to integrate with transfer services. With additional constraints, LinTS can be extended for spatiotemporal scheduling, and multi-objective solvers can be used when there are tradeoffs in the objective function.

Acknowledgements

This project is in part sponsored by the National Science Foundation (NSF) under award number OAC-2313061. We also thank Chameleon Cloud for making their resources available for the experiments of this work.

References

[1] Carly Davenport et al. *Generational Growth: AI data centers and the coming US power surge*. Goldman Sachs, Apr. 2024. URL: https://www.goldmansachs.com/pdfs/insights/pages/generational-growth-ai-data-centers-and-the-coming-us-power-surge/report.pdf.

[2] *Google Environmental Report 2024*. 2024. URL: https://sustainability.google/reports/google-2024-environmental-report.

[3] *Meta 2024 Sustainability Report*. 2024. URL: https://sustainability.atmeta.com/2024-sustainability-report.

[4] *2024 Environmental Sustainability Report*. 2024. URL: https://www.microsoft.com/en-us/corporate-responsibility/sustainability/report.

[5] Michael Terrell. *New nuclear clean energy agreement with Kairos Power*. Oct. 2024. URL: https://blog.google/outreach-initiatives/sustainability/google-kairos-power-nuclear-energy-agreement.

[6] Bilge Acun et al. "Carbon Explorer: A Holistic Framework for Designing Carbon Aware Datacenters". In: *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ASPLOS 2023. Vancouver, BC, Canada: Association for Computing Machinery, 2023, pp. 118–132.

[7] Thanathorn Sukprasert et al. "On the Limitations of Carbon-Aware Temporal and Spatial Workload Shifting in the Cloud". In: *Proceedings EuroSys'24*. Athens: Association for Computing Machinery, 2024, pp. 924–941.

[8] Anshul Gandhi et al. "Metrics for Sustainability in Data Centers". In: *ACM SIGENERGY Energy Informatics Review* 3 (2023), pp. 40–46. URL: https://api.semanticscholar.org/CorpusID:251304999.

[9] Sirui Qi et al. *MOSAIC: A Multi-Objective Optimization Framework for Sustainable Datacenter Management*. 2023. arXiv: 2311.08583 [cs.DC].

[10] Thomas Anderson et al. "Treehouse: A Case For Carbon-Aware Datacenter Software". In: *SIGENERGY Energy Inform. Rev.* 3.3 (Oct. 2023), pp. 64–70.

[11] Ana Radovanović et al. "Carbon-Aware Computing for Datacenters". In: *IEEE Transactions on Power Systems* 38.2 (Mar. 2023), pp. 1270–1280. ISSN: 1558-0679. DOI: 10.1109/TPWRS.2022.3173250.

[12] Jr. Thomas Barnett et al. *Cisco Global Cloud Index 2015–2020*. URL: https://www.cisco.com/c/dam/m/en_us/service-provider/ciscoknowledgenetwork/files/622_11_15-16-Cisco_GCI_CKN_2015-2020_AMER_EMEAR_NOV2016.pdf.

[13] Malgorzata Wiatros-Motyka et al. *Global Electricity Review 2023 — Ember — ember-energy.org*. https://ember-energy.org/latest-insights/global-electricity-review-2023/. [Accessed 30-03-2025]. 2023.

[14] Clarisse Aujoux, Kumiko Kotera, and Odile Blanchard. "Estimating the carbon footprint of the GRAND project, a multi-decade astrophysics experiment". In: *Astroparticle Physics* 131 (2021), p. 102587.

[15] *Electricity Maps*. 2024. URL: https://www.electricitymaps.com.

[16] *WattTime*. 2024. URL: https://watttime.org.

[17] Walid A. Hanafy et al. "CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency". In: *Proc. ACM Meas. Anal. Comput. Syst.* 7.3 (Dec. 2023).

[18] Yuchao Zhang et al. "BDS+: An Inter-Datacenter Data Replication System With Dynamic Bandwidth Separation". In: *IEEE/ACM Transactions on Networking* 29.2 (2021), pp. 918–934. DOI: 10.1109/TNET.2021.3054924.

[19] Zichen Xu et al. "CADRE: Carbon-Aware Data Replication for Geo-Diverse Services". In: *2015 IEEE International Conference on Autonomic Computing*. 2015, pp. 177–186. DOI: 10.1109/ICAC.2015.15.

[20] Jeonghyeon Park et al. "Carbon-Aware and Fault-Tolerant Migration of Deep Learning Workloads in

the Geo-Distributed Cloud". In: *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. 2024, pp. 494–501.

[21] Adam Lechowicz et al. "CarbonClipper: Optimal Algorithms for Carbon-Aware Spatiotemporal Workload Management". In: 2024. URL: https://api.semanticscholar.org/CorpusID:271874702.

[22] John Thiede et al. "Carbon Containers: A System-level Facility for Managing Application-level Carbon Emissions". In: *Proceedings of the 2023 ACM Symposium on Cloud Computing*. SoCC '23. Santa Cruz: Association for Computing Machinery, 2023, pp. 17–31.

[23] Sophie Hall et al. *Carbon-Aware Computing for Data Centers with Probabilistic Performance Guarantees*. 2024. arXiv: 2410.21510 [eess.SY]. URL: https://arxiv.org/abs/2410.21510.

[24] Philipp Wiesner et al. "Cucumber: Renewable-Aware Admission Control for Delay-Tolerant Cloud and Edge Workloads". In: *European Conference on Parallel Processing*. 2022. URL: https://doi.org/10.1007/978-3-031-12597-3_14.

[25] Sirui Qi et al. "CASA: A Framework for SLO and Carbon-Aware Autoscaling and Scheduling in Serverless Cloud Computing". In: *ArXiv* abs/2409.00550 (2024). URL: https://arxiv.org/abs/2409.00550.

[26] Philipp Wiesner et al. "Let's wait awhile: how temporal workload shifting can reduce carbon emissions in the cloud". In: *Proceedings of the 22nd International Middleware Conference* (2021). URL: https://doi.org/10.1145/3464298.3493399.

[27] Soumyendu Sarkar et al. "Real-time Carbon Footprint Minimization in Sustainable Data Centers with Reinforcement Learning". In: *NeurIPS 2023 Workshop on Tackling Climate Change with Machine Learning*. 2023.

[28] Dongxiang Yan, Mo-Yuen Chow, and Yue Chen. "Low-Carbon Operation of Data Centers With Joint Workload Sharing and Carbon Allowance Trading". In: *IEEE Transactions on Cloud Computing* 12 (2024), pp. 750–761. URL: https://doi.org/10.1109/TCC.2024.3396476.

[29] *perf: Linux profiling with performance counters — perfwiki.github.io*. https://perfwiki.github.io/main/. [Accessed 28-03-2025].

[30] *Intel® 64 and IA-32 Architectures Software Developer Manuals — intel.com*. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html. [Accessed 25-03-2025].

[31] *NVML API Reference Guide*. Nvidia. Nvidia, 2025. URL: https://docs.nvidia.com/deploy/pdf/NVML_API_Reference_Guide.pdf.

[32] Saurabhsingh Rajput et al. "Enhancing Energy-Awareness in Deep Learning through Fine-Grained Energy Measurement". In: *ACM Trans. Softw. Eng. Methodol.* 33.8 (Dec. 2024). ISSN: 1049-331X. DOI: 10.1145/3680470. URL: https://doi.org/10.1145/3680470.

[33] *GitHub - green-kernel/powerletrics: Powermetrics for Linux — github.com*. https://github.com/green-kernel/powerletrics. [Accessed 28-03-2025]. Green Kernel.

[34] Jaylen Wang, Udit Gupta, and Sriraman Akshitha. "Peeling Back the Carbon Curtain: Carbon Optimization Challenges in Cloud Computing". In: *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (2023). URL: https://api.semanticscholar.org/CorpusID:260380570.

[35] Walid A. Hanafy et al. "Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ASPLOS '24. La Jolla, CA, USA: Association for Computing Machinery, 2024, pp. 479–496.

[36] Abel Souza et al. "Ecovisor: A Virtual Energy System for Carbon-Efficient Applications". In: *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ASPLOS 2023. Vancouver, BC, Canada: Association for Computing Machinery, 2023, pp. 252–265.

[37] Engin Arslan and Tevfik Kosar. "High-speed transfer optimization based on historical analysis and real-time tuning". In: *IEEE Transactions on Parallel and Distributed Systems* 29.6 (2018), pp. 1303–1316.

[38] Tevfik Kosar. *Data placement in widely distributed systems*. The University of Wisconsin-Madison, 2005.

[39] George Kola et al. "DISC: A System for Distributed Data Intensive Scientific Computing." In: *WORLDS*. 2004.

[40] Mohak Chadha et al. "GreenCourier: Carbon-Aware Scheduling for Serverless Functions". In: *Proceedings of the 9th International Workshop on Serverless Computing* (2023). URL: https://dl.acm.org/doi/10.1145/3631295.3631396.

[41] Viktor Gsteiger et al. "Caribou: Fine-Grained Geospatial Shifting of Serverless Applications for Sustainability". In: URL: https://api.semanticscholar.org/CorpusID:273692349.

[42] Mohammad Aldossary and Hatem A. Alharbi. "Towards a Green Approach for Minimizing Carbon Emissions in Fog-Cloud Architecture". In: *IEEE Access* 9 (2021), pp. 131720–131732. DOI: 10.1109/ACCESS.2021.3114514.

[43] Li Wu et al. *CarbonEdge: Leveraging Mesoscale Spatial Carbon-Intensity Variations for Low Carbon Edge Computing*. 2025. arXiv: 2502.14076 [cs.DC]. URL: https://arxiv.org/abs/2502.14076.

[44] Vašek Chvátal. *Linear Programming*. en. Macmillan, Sept. 15, 1983. ISBN: 9780716715870. URL: https://books.google.com/books/about/Linear_Programming.html?hl=&id=DN20_tW_BV0C.

[45] Ismail Alan, Engin Arslan, and Tevfik Kosar. "Energy-aware data transfer algorithms". In: *Proceedings of*

the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '15. Austin, Texas: Association for Computing Machinery, 2015.

[46] Kate Keahey et al. "Lessons Learned from the Chameleon Testbed". In: *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[47] *linprog; SciPy v1.15.2 Manual — docs.scipy.org*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html. [Accessed 28-03-2025].

[48] Roozbeh Bostandoost et al. "Data-driven Algorithm Selection for Carbon-Aware Scheduling". In: *Proceedings of the 3rd Workshop on Sustainable Computer Systems. HotCarbon*. Vol. 24. 2024.

[49] Vivien Gueant. *iPerf - The TCP, UDP and SCTP network bandwidth measurement tool — iperf.fr*. https://iperf.fr/. [Accessed 28-03-2025].