RESEARCH-ARTICLE

# GridGreen: Integrating Serverless Computing in HPC Systems for Performance and Sustainability

AMIT SAMANTA, The University of Utah, Salt Lake City, UT, United States

RYAN STUTSMAN, The University of Utah, Salt Lake City, UT, United States

ROHAN BASU ROY, The University of Utah, Salt Lake City, UT, United States

Open Access Support provided by:

The University of Utah

**PDF Download**
**3772052.3772241.pdf**
**26 January 2026**
**Total Citations:** 0
**Total Downloads:** 52

**Published:** 19 November 2025

**Citation in BibTeX format**

SoCC '25: ACM Symposium on Cloud Computing
*November 19 - 21, 2025*
*Online, USA*

**Conference Sponsors:**
SIGOPS
SIGMOD

# GridGreen: Integrating Serverless Computing in HPC Systems for Performance and Sustainability

Amit Samanta
University of Utah
Salt Lake City, USA

Ryan Stutsman
University of Utah
Salt Lake City, USA

Rohan Basu Roy
University of Utah
Salt Lake City, USA

## Abstract

*We present GridGreen, a scheduling framework that improves the sustainability and performance of scientific workflow execution by integrating serverless computing with traditional on-premise high performance computing (HPC) clusters. GridGreen allocates workflow components across HPC and serverless environments leveraging spatio-temporal variation of carbon intensity and component execution characteristics. It incorporates component-level optimization, speculative pre-warming, I/O-aware data management, and fallback adaptation to jointly minimize carbon footprint and service time under user-defined cost constraints. Our evaluations on large-scale bioinformatics workflows across leadership-class HPC facilities and cloud-based serverless regions demonstrate that GridGreen achieves robust, cost-effective execution while improving carbon efficiency.*

## CCS Concepts

• **Computer systems organization → Cloud computing**.

## Keywords

Cloud Computing, High Performance Computing, Serverless Computing, Carbon Footprint, Sustainability.

**ACM Reference Format:**
Amit Samanta, Ryan Stutsman, and Rohan Basu Roy. 2025. GridGreen: Integrating Serverless Computing in HPC Systems for Performance and Sustainability . In *ACM Symposium on Cloud Computing (SoCC '25), November 19–21, 2025, Online, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3772052.3772241

## 1 Introduction

**Why is sustainability of HPC systems becoming a growing concern?** High performance computing (HPC) systems are becoming central to progress in scientific discovery, engineering, and national infrastructure. However, the growing scale and usage intensity of these systems have raised serious concerns about their environmental sustainability [38]. As compute demands surge — from global data generation projected to grow from 1.2 trillion gigabytes in 2010 to 175 trillion by 2025 [18] — HPC facilities continue to scale up, often at the cost of energy efficiency. For instance, the Aurora supercomputer (2025) of Argonne Leadership Computing Facility consumes more than 30× the peak power of Theta (2017) [52]. Power consumption alone does not fully capture sustainability: the environmental impact also depends on the carbon intensity (CI) of the regional power grid and the embodied carbon from manufacturing and deploying HPC hardware [16, 28]. While many datacenters aim to offset emissions [3, 6, 23, 47], current projections estimate that data and HPC systems could account for up to 8% of global emissions by 2030 without mitigation [4]. This motivates a shift toward carbon-aware HPC design and workload scheduling to address sustainability alongside performance. Recent efforts [2, 15, 25, 40] have started to explore this space, but effective solutions must jointly account for system-level heterogeneity, workload characteristics, and real-time carbon intensity variability.

**How can we improve the sustainability of production HPC systems?** Traditional HPC systems are deployed in fixed physical locations with dedicated hardware, which makes their carbon impact largely dependent on two factors: the embodied emissions of the installed infrastructure and the carbon intensity (CI) of the local power grid impacting the operational emissions. Efforts to improve sustainability in this model often focus on optimizing code to use fewer resources or reduce energy consumption per job [65, 65, 74]. Several HPC centers have started incentivizing such practices by offering carbon credits or queue priority for energy-efficient jobs [34, 62, 71]. However, these strategies are inherently limited by the constraints of the underlying HPC architecture, which is tied to a single location (thus, limiting operational emission reduction opportunities) and fixed hardware configuration (thus, limiting embodied emission reduction opportunities).

In contrast, serverless computing, already gaining traction in HPC for autoscaling, modularization, and resource efficiency [20, 55, 56, 58, 59], offers new opportunities for sustainable workload execution. Serverless functions can be executed across multiple cloud regions, and this geographical flexibility enables leveraging spatio-temporal variations in CI. When CI is high at the HPC facility, components can be offloaded as stateless serverless functions to cleaner regions, which is not possible in conventional HPC systems. Moreover, serverless platforms can also improve embodied carbon footprint by introducing hardware heterogeneity. While serverless has been explored for performance and elasticity [11], its potential for improving the sustainability of HPC remains unexplored.

**What are the challenges of integrating serverless into HPC?** While serverless computing offers flexibility, scalability, and potential sustainability benefits, integrating it into HPC workflows introduces several challenges. Serverless platforms are stateless and lack direct communication primitives, leading to I/O overhead when components exchange data across phases [10]. Cold start latency [35], unpredictable execution environments, and the absence of energy visibility further complicate scheduling decisions [32]. Moreover, serverless computing can be expensive if not used judiciously, particularly when functions are invoked frequently or involve large data transfers [57]. From a sustainability standpoint,

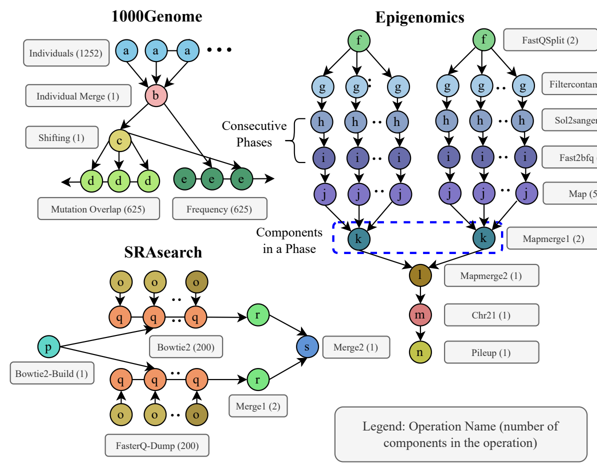


**Figure 1: HPC applications are being designed as workflows with multiple phases and with components in each phase. The components in the same phase can run in parallel, and two consecutive phases run one after another.**

offloading to serverless without accounting for dynamic carbon intensity, data movement, or embodied emissions can lead to suboptimal outcomes [54]. Addressing these challenges requires a coordinated system that accounts for performance, cost, and carbon.

**Key Contributions.** The following are our key contributions:

1. We present the first system for hybrid execution of HPC workflows that integrates traditional HPC infrastructure with production-grade serverless platforms to improve sustainability. GridGreen jointly minimizes carbon footprint and service time under a user-defined cost constraint
2. We improve the performance of widely used bioinformatics workflows – 1000Genome, Epigenomics, and SRAsearch [13, 45, 60].
3. We design and implement a scheduling framework that incorporates component-level optimization, speculative pre-warming, I/O-aware data management, and fallback adaptation, guided by real-time carbon intensity and profiling data.
4. We demonstrate that GridGreen achieves balanced performance and carbon footprint across varying cost constraints, serverless regions, and HPC facilities including Argonne, Lawrence Berkeley, and Oak Ridge, leveraging spatio-temporal carbon variation. GridGreen is open-sourced at: https://doi.org/10.5281/zenodo.15244027.

## 2 Background and Motivation

In this section, we discuss the necessary background details and motivate the design of GridGreen.

**HPC Workflows and Execution Model.** Scientific and HPC applications, used in domains such as biology, physics, and climate modeling, are increasingly following a modular workflow-based design, structured as Directed Acyclic Graphs (DAGs) [5, 12, 22, 41, 49, 51]. Each node in the DAG represents a distinct computational *component* (a collection of component can together perform an operation), and edges encode data or control dependencies. This decomposition improves scalability, encourages reuse, and enables better performance tuning compared to monolithic applications.

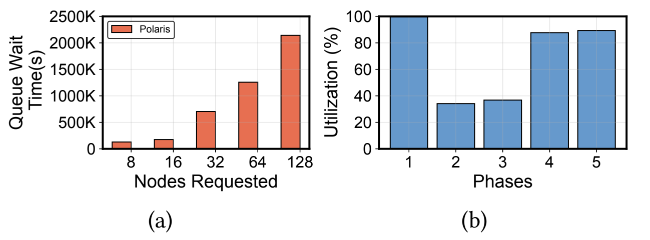


**Figure 2: Queue wait time significantly impacts service time (a), and HPC workflows underutilize allocated nodes (b).**

We study three representative large-scale scientific workflows: **1000Genome**, **SRAsearch**, and **Epigenomics** [13, 45, 60]. These workflows automate critical genomic and biomedical processing pipelines. They are all open-sourced and largely capture characteristics of production HPC workflows. For example, more than 80% of components of these workflows are used in other major Exascale Computing Projects (EXAALT, GAMESS, and ExaAM), belonging to various domains (material-science, biology, and electromagnetics) [46]. Each workflow consists of multiple *phases*, where a phase is a group of components that can execute in parallel. Two consecutive phases execute sequentially, and dependencies across components – such as fan-in, fan-out, and many-to-many data transfers, resulting in significant data movement between phases. The workflows contain multiple phases with hundreds of components in each phase, as shown in Fig. 1. The number of components and their resource demands vary widely across phases, resulting in different compute resource requirements at different phases of execution.

In traditional HPC environments, workflows are submitted through batch scheduling systems such as SLURM or PBS [8, 75]. A user specifies the number of nodes required and the workflow is queued until resources become available. This *queue wait time* is non-trivial and can significantly impact overall workflow's *service time* (execution time plus starting time, which is the queue wait time in this case). Fig. 2(a) shows the average queue wait time for jobs with different numbers of requested nodes of the supercomputer Polaris at Argonne Leadership Computing Facility. We see that the queue wait time is significant and generally increases with the number of requested nodes, as larger dedicated jobs are more difficult to schedule within the cluster's fixed resource pool.

Once allocated, HPC nodes remain dedicated to a workflow for the entire duration of its execution. However, as illustrated in Fig. 1, the number of components can vary significantly across workflow phases. In addition, different phases may have distinct resource requirements depending on the nature and scale of their components. Since HPC environments require users to pre-allocate a fixed number of nodes, this leads to resource under- or over-utilization across phases. In practice, users tend to allocate enough resources to handle the most resource-intensive phase, which results in substantial underutilization during other phases. Fig. 2(b) demonstrates this effect for the SRASearch workflow, showing low utilization across most phases when 200 nodes are allocated for execution. This approach can also increase the overall service time, as requesting more nodes typically leads to longer queue wait times.

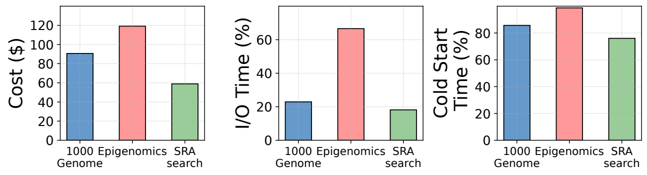


**Figure 3: Serverless computing can have high execution costs, I/O time, and cold start latencies (I/O Time and Cold Start Time are expressed as a percentage of service time).**

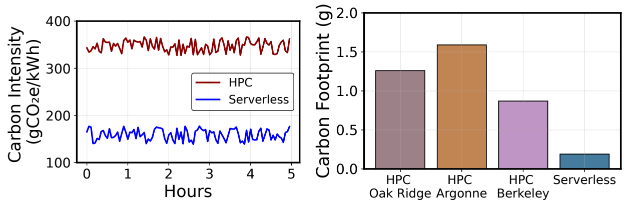
**Observation 1.** *While HPC workflows offer modularity and parallelism through their phase-component structure, different phases often have varying resource requirements. In traditional HPC settings with fixed node allocation, this mismatch leads to resource underutilization across phases and longer queue wait times when over-provisioning for the most demanding phase.*

**How can serverless computing be effective for HPC?** Serverless computing is a cloud service model where users deploy small, stateless functions that automatically scale to handle demand and are billed only for actual execution time. This model eliminates the need for managing underlying infrastructure and provides fine-grained resource allocation. Functions scale automatically and are billed on a pay-as-you-go basis, making the model flexible and cost-efficient for bursty workloads. This autoscaling capability directly addresses the resource under- and over-utilization challenges discussed earlier in HPC workflows, where phases with varying component counts and compute demands are executed over a fixed set of pre-allocated nodes. In a hybrid setting – combining serverless with a set of HPC nodes, several components can be offloaded as functions, reducing the need for large HPC reservations (thus potentially lowering total service time due to the reduction of queue wait time) and improving the HPC system utilizations. Due to this flexibility, leadership HPC centers have started integrating serverless into their platforms [11].

However, while serverless introduces benefits, it also presents new challenges in HPC contexts. *First*, production serverless environments can incur significant costs. Fig. 3 shows that executing a single run of our target workflows purely via serverless can cost over hundred dollars, and the cost grows even more with an increase in the input size. In practice, these workflows are run repeatedly by users with varying inputs and configurations, amplifying the cumulative expense of full serverless execution.

*Second*, serverless functions are stateless and lack direct communication mechanisms. In public cloud environments, functions do not expose network identities and cannot communicate via message passing. Instead, data is exchanged through external storage systems (e.g., S3 or EFS in AWS), which introduces significant latency. HPC workflows are often I/O-intensive, and such storage-mediated transfers can constitute a large fraction of total service time – exceeding 60% in some cases, as seen in Fig. 3. Thus, naively offloading all components can degrade performance unless I/O-sensitive components are carefully placed or data exchange is explicitly managed.

*Third*, serverless functions suffer from *cold start* delays. When a function is invoked, its code, dependencies, and input data must be transferred to the server and loaded into memory. This initialization



(a) Carbon Intensity (b) Carbon Footprint

**Figure 4: Serverless execution has the potential to reduce the effective carbon footprint over traditional HPC execution.**

time can dominate execution, especially for workflows with large input and dependency sizes. To mitigate the effects of cold starts, service providers keep-alive [57] or pre-warm [48] functions, which basically means having the function and its dependencies loaded into memory on a server ahead of invocation, so that subsequent requests can have a *warm start* (avoid cold start overhead). However, keep-alive or pre-warming consumes resources and is costly for a provider. Cloud providers must pre-allocate resources to keep a function warm, but predicting whether a pre-warmed function will actually be invoked is challenging – otherwise, the pre-warming effort results in wasted resources.

As shown in Fig. 3, the cold start time can be a significant percentage of the total *service time* (execution time including I/O time and cold start time, which is zero in case of a warm start) if all components of a workflow experience cold starts – sometimes the service time can even double compared to execution time, as seen for Epigenomics. This overhead arises because serverless platforms need to provision runtime environments on demand, load code, and initialize dependencies before execution begins. Therefore, to mitigate this effect, components must be selectively and opportunistically pre-warmed before invocation.

**Observation 2.** *Serverless offers autoscaling and pay-as-you-go benefits that can complement HPC by reducing underutilization and queue delays. However, challenges such as high cost, I/O bottlenecks, and cold starts necessitate intelligent scheduling and selective pre-warming to maintain performance and efficiency.*

**How can serverless computing make HPC sustainable?** We discussed how serverless computing offers benefits for HPC in terms of resource utilization and queue wait time improvement. However, another less explored but increasingly urgent motivation for considering serverless computing in HPC is environmental sustainability. As leadership-scale systems continue to grow in power consumption and usage scale, the carbon footprint associated with HPC workloads has become a growing concern [38].

Carbon emissions are typically categorized into two components: embodied and operational. *Embodied carbon footprint* refers to the emissions associated with manufacturing, transporting, and decomposing computing hardware – this is considered a one-time cost, amortized over the lifetime of hardware. *Operational carbon footprint* is the carbon emissions during the operation of a hardware and it is the product of energy consumption and *carbon intensity* (CI). CI represents the amount of carbon emitted per kilowatt-hour of electricity consumed and varies depending on the mix of energy sources in a region's grid. Power generated from renewable sources (e.g., wind or hydro) generally results in a much lower CI compared to fossil-fuel-based sources (e.g., coal or natural gas). Importantly, CI is not static; it varies both spatially and temporally. However, a traditional HPC system is bound to a fixed location, meaning it is unable to adapt to carbon intensity fluctuations in other regions. Serverless computing, by contrast, is built on stateless functions that can be launched in multiple production cloud regions across the world. This flexibility provides an opportunity to dynamically shift execution to cloud regions with lower CI at any given point in time to leverage the spatio-temporal variation of CI.

To illustrate this potential, we conduct a simple experiment using a 5-hour run of the 1000Genome workflow. In one configuration, the entire workflow is executed on an HPC center located near Chicago (Argonne). In a second configuration, the same workflow components are offloaded to whichever serverless region – among six geographically distributed locations (California, Virginia, Ireland, Germany, Korea, and Brazil), has the lowest CI at each point of time during execution of components in different phases. Fig.4(a) shows that the dynamic serverless configuration significantly lowers resultant CI, associated with the workflow execution based on the region of execution, compared to static execution in a fixed HPC location. In Fig.4(b), we comparing the total carbon footprint (dominated by operational emissions) for executing the workflow on three different HPC sites (Argonne, Oak Ridge, and Lawrence Berkeley) versus purely via serverless execution across the six aforementioned cloud regions. We observe that serverless execution lowers the carbon footprint, owing to its ability to shift computation to cleaner grids in real-time. These findings suggest that serverless can complement HPC to improve the sustainability of HPC workflow executions. Realizing these benefits, however, requires solving scheduling challenges such as identifying which workflow components to shift, when and where to shift them, and how to handle cold start overheads and data movement across regions.

**Observation 3.** *Serverless enables HPC workflows to reduce carbon footprint by shifting execution to regions with lower carbon intensity. Unlike fixed-location HPC, it leverages spatio-temporal CI variation, though effective use requires careful scheduling.*

## 3 Design and Implementation

In this section, we first discuss GridGreen's objectives and design overview and then provide the design and implementation details.

### 3.1 Objectives and Overview

GridGreen is designed to jointly minimize (a) total service time and (b) carbon footprint of executing HPC workflows, expressed as dynamic acyclic graphs (DAGs), while adhering to a strict cost budget. To achieve this, GridGreen strategically allocates resources and schedules DAG components across heterogeneous compute environments – a home HPC facility and serverless cloud platforms.

Upon submission of a workflow to the home HPC facility, GridGreen first determines the optimal number of nodes to allocate locally. Once the home HPC facility resources are available and execution begins, GridGreen dynamically decides which components to execute in the local HPC environment and which to offload to serverless cloud infrastructure. For offloaded components, GridGreen selects cloud datacenter regions that best balances carbon efficiency and service time within cost constraints.

To mitigate serverless-specific overheads, GridGreen employs speculative function pre-warming to reduce cold starts and implements intelligent data and I/O coordination mechanisms to manage the movement of intermediate data between the HPC system and cloud endpoints. Additionally, when predictions about the carbon intensity of datacenter locations become inaccurate, GridGreen activates fallback mechanisms to preserve performance and carbon efficiency. Fig. 5 provides an overview of GridGreen's operation. Overall, GridGreen is a robust, cost-aware orchestration framework that adapts to dynamic execution conditions to deliver sustainable, and high-performant execution of HPC workflows.

### 3.2 GridGreen's Design Components

We now describe the individual components that makeup GridGreen's design. Each component is responsible for a distinct purpose: queue wait time and carbon intensity prediction guide the initial HPC node allocation; the scheduler assigns components to HPC or serverless based on execution profiles, transfer overheads, and cost constraints; pre-warming minimizes serverless cold-start latency; the data management layer handles inter-component transfers across execution environments; and the fallback mechanism adjusts assignments when queue load or carbon intensity deviates significantly from predictions. These components are integrated to ensure that scheduling and execution decisions remain jointly optimized for service time, and carbon, under a budget constraint.

**HPC Node Allocation with Cost and Carbon Predictions.** When a workflow is submitted at the home HPC facility, GridGreen determines how many nodes $N$ to request in the HPC cluster. Unlike a purely performance-driven approach, GridGreen factors in (a) projected queue wait time $Q(N)$ at the HPC facility, (b) potential serverless costs and carbon intensities, and (c) a global cost budget $B$. To accomplish this, GridGreen considers a feasible range $N \in \{N_{\min}, \ldots, N_{\max}\}$ and predicts the queue wait time $Q(N)$ via a regression model trained on the past one month of job logs. This captures how requesting more nodes often leads to higher queue delays due to resource contention and scheduling backlogs [42].

In parallel, GridGreen forecasts near-future carbon intensity (CI) both for the home HPC facility and for each serverless region using short-horizon prediction models aligned with WattTime's methodology [1, 73]. WattTime is widely used for real-time and forecasted emissions monitoring in systems research [30, 31, 37, 63]. It collects data from grid operators, U.S. EPA, NOAA, and EIA. It computes marginal carbon intensity forecasts by combining historical grid emission data with real-time market dispatch information from independent system operators (ISOs). Its prediction model incorporates a rolling window of past emissions (typically the past 24–72 hours) and grid mix characteristics, then applies a gradient-boosted regression tree ensemble to forecast CI over a 24-hour horizon in 5-minute intervals [44, 53, 66]. GridGreen uses WattTime's CI prediction – for each datacenter region $r \in \{1, \ldots, R\}$ and the local HPC facility's region $CI_{hpc}(t)$, it constructs a model that takes as input historical CI traces, grid generation type shares (e.g., coal, solar, wind), and local demand patterns to predict $CI_r(t)$ for future

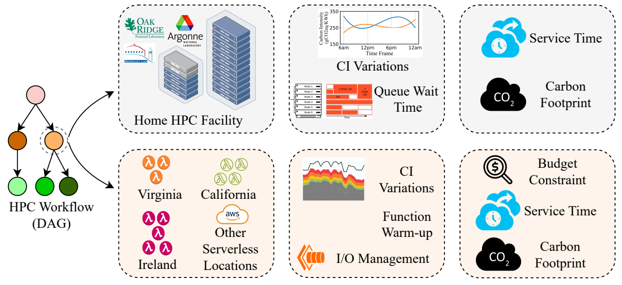


**Figure 5: GridGreen allocates resources in a home HPC facility, and dynamically schedules components in HPC facility and across various serverless locations to jointly optimize service time and carbon footprint, under a budget constraint.**

times $t$ corresponding to expected execution phases. This prediction enables GridGreen to determine how CI will evolve over the hours during which the workflow runs, and hence influences both initial node allocation and downstream scheduling decisions.

Given these predictions and the global cost budget $B$, GridGreen enumerates candidate values of $N$ and assesses whether the resulting queue delay plus required serverless offloading (under a cost constraint) can likely yield acceptable overall service time and carbon footprint. In other words, if the system anticipates limited benefit from large $N$ due to extremely long queue times, it may favor a smaller allocation combined with heavier use of serverless (within the budget). Conversely, if local HPC nodes are cleaner or if serverless usage might exceed cost constraints, GridGreen can reserve more nodes. *This interplay of queue wait, cost, and future carbon profiles drives the initial decision of* $N$. Next, we discuss GridGreen's component-level scheduling approach.

**Component-Level Scheduling and Optimization.** Once GridGreen selects $N$ HPC nodes and they become available, workflow execution begins. HPC workflows typically execute in multiple sequential phases, each consisting of components that run in parallel. Components within a phase are assigned to execute either on a single HPC node or on serverless functions in one of several cloud regions. The assignment must respect the global cost budget of $B$, which accounts for all serverless execution and data transfer costs. Components running on HPC nodes do not incur monetary costs.

To determine the schedule for optimizing service time and carbon footprint, GridGreen performs a brute-force simulation search over possible component assignments to HPC and serverless locations. For each component, GridGreen leverages pre-profiled execution time and energy measurements obtained separately for a single HPC node and for each potential serverless region. These profiles, obtained periodically and reused across multiple workflow runs, introduce negligible overhead due to the relatively static nature of hardware platforms. During profiling, the execution time of each task is computed as the sum of its profiling-based execution time and any associated data transfer delays if its input is produced in a different execution environment (HPC facility or another region).

Within each workflow phase, since components execute concurrently, the phase execution time is defined as the maximum runtime across all components in that phase. Formally, for a phase $p$ with component set $\mathcal{V}_p$, the execution time is:

$$T_p = \max_{v \in \mathcal{V}_p} \left( t_v^{exec} + t_v^{xfer} \right) \tag{1}$$

where $t_v^{exec}$ is the execution time of component $v$ in its assigned location (HPC or serverless), and $t_v^{xfer}$ is the additional time required to transfer input data for component $v$ if produced in a different environment. Given sequential phase execution, the total workflow service time is then calculated as:

$$T_{\text{total}} = Q(N) + \sum_{p=1}^{P} T_p \tag{2}$$

where $Q(N)$ is the initial queue wait time at the HPC facility for allocating $N$ nodes, and $P$ is the total number of workflow phases.

To estimate the total carbon footprint, GridGreen considers both operational and embodied carbon emissions. Operational carbon is computed as the product of each component's measured energy consumption (from profiling) and the forecasted carbon intensity (CI) at its assigned location during execution. Embodied carbon is added to reflect the emissions associated with manufacturing hardware. Thus, the total carbon footprint is given by:

$$C_{\text{total}} = \sum_{p=1}^{P} \sum_{v \in \mathcal{V}_p} \left( E_v \times CI_v(t_v) + C_v^{emb} \right) \tag{3}$$

where $E_v$ denotes the operational energy usage of component $v$, $CI_v(t_v)$ represents the forecasted carbon intensity at the assigned location and time, and $C_v^{emb}$ is the amortized embodied carbon for the hardware running $v$. In our execution environment, components are executed either on uniform HPC nodes or on similar serverless hardware provided by a single cloud vendor – this difference in execution environment causes variations in embodied carbon across assignments. However, we observe that the impact of operational carbon is greater, and considering only operational carbon would yield nearly identical outcomes. Nevertheless, for completeness, GridGreen explicitly accounts for embodied carbon also.

Through exhaustive search via simulation, GridGreen optimizes scheduling by evaluating every feasible component assignment across HPC nodes and multiple serverless regions, subject to the global cost constraint $B$. Specifically, the optimization jointly minimizes total service time and total carbon footprint (with equal or other configurable weights on both objectives), enumerating choices across execution environments and regions. The exhaustive search completes in under a minute for all evaluated workflows, while the workflow execution times are often in the range of hours. The profiling cost, both in time and carbon, is amortized over 5 to 7 runs of the same workflow, while developed HPC workflows are executed hundreds of times. This combination of exhaustive search and upfront profiling is also used in other HPC-cloud scheduling solutions due to its negligible overhead relative to long workflow runtimes, and the resulting gains outweigh the profiling costs since they are executed repeatedly across multiple submissions [10, 55].

**Speculative Pre-Warming in Serverless Regions.** To mitigate cold-start delays in serverless platforms, GridGreen employs speculative *pre-warming*. At the end of each phase, when execution is nearing completion, GridGreen speculatively determines which components are likely to be offloaded in the next phase based on their assignment from the scheduling step. It identifies the serverless region with the lowest forecasted carbon intensity at that time and selects it as the target for pre-warming. For each such component, GridGreen triggers a *dummy function invocation* in the selected region – this involves deploying the actual function executable bundled with its data and dependencies as a zipped package, invoking it, and immediately terminating the function. On AWS Lambda, this dummy invocation causes the backend to send the zipped package to a node in the selected region, unpack it, load dependencies, and cache them in memory, effectively initializing the container and keeping it warm for a short interval [70].

Our pre-warming strategy is deterministic and robust by design. For a given workflow and its scheduling assignment, pre-warming decisions are uniquely determined by the mapping of components to HPC or serverless regions and by predicted carbon intensities. At the end of each phase, only components in the next phase assigned to serverless regions are pre-warmed, and each such component is pre-warmed exactly once in the region with the lowest forecasted carbon intensity. This deterministic approach ensures consistent behavior across executions and avoids redundant warm-ups. Additionally, if pre-warming fails or predicted conditions change, GridGreen's fallback mechanism (discussed later) dynamically reassigns components to alternate regions to maintain reliability.

By pre-warming in the chosen low-carbon region, GridGreen ensures that when a function is used to invoke a component in the next phase, it has a warm start, avoiding the cold start overhead. In some of our evaluated workflows, cold starts without pre-warming led to 8–13% increases in overall phase execution time, particularly for components with large dependencies and datasets. The cost of pre-warming is minimal and is accounted toward the total cost constraint, but its service time benefits outweigh this overhead, making it an important component in GridGreen's design.

**Data Management and Transfer Costs.** Data movement between components affects both service time and cost. Most data transfers occur at phase boundaries, as outputs of one phase become inputs to the next. When components are distributed across the HPC facility and serverless regions, GridGreen accounts for three primary kinds of transfers: (a) HPC facility to serverless region, (b) serverless to serverless within the same region, and (c) serverless in one region to serverless in another region.

Transfers from HPC to serverless involve staging output data from the HPC file system to cloud storage in the destination region, which is then accessed by the receiving serverless function. If components within a phase are co-located in a single serverless region, they share a common cloud-backed file system (e.g., AWS Elastic File System) mounted to each function. Serverless-to-serverless communication across regions, however, requires copying intermediate data to remote object storage and reloading it in the next region – a process that adds latency and incurs cross-region egress fees. Serverless functions do not support direct inter-function communication (e.g., via sockets or MPI), so all data exchange must be mediated via file I/O on shared storage.

For workflows with components originally designed for tightly coupled HPC execution using MPI, GridGreen translates inter-component MPI exchanges to file-based coordination using remote

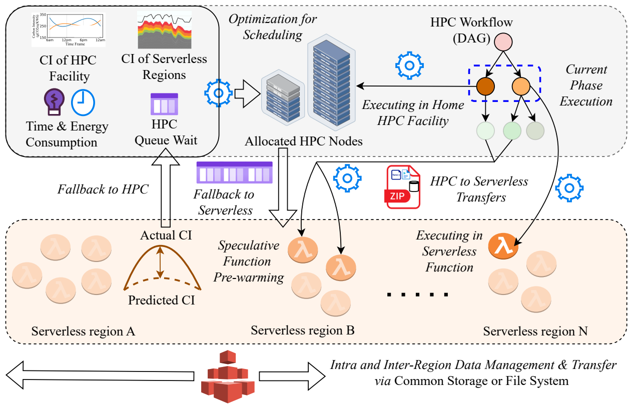

CI of HPC Facility
CI of Serverless Regions
Time & Energy Consumption
HPC Queue Wait
Optimization for Scheduling
Allocated HPC Nodes
HPC Workflow (DAG)
Executing in Home HPC Facility
Current Phase Execution
Fallback to HPC
Fallback to Serverless
HPC to Serverless Transfers
Actual CI
Predicted CI
Speculative Function Pre-warming
Executing in Serverless Function
Serverless region A
Serverless region B
Serverless region N
Intra and Inter-Region Data Management & Transfer via Common Storage or File System

**Figure 6: GridGreen integrates component-level scheduling, dynamic HPC node allocation, carbon- and service-time-aware optimization, speculative pre-warming, I/O and data management, and fallback mechanisms in its design.**

storage mounts. Each MPI send/receive pair is rewritten as a two-step process: writing data to a shared file system from the sender and loading it from storage by the receiver. While this model introduces additional latency, it ensures correctness and portability across regions. These latencies are explicitly measured and included in the component's transfer delay $t_v^{xfer}$ and overall execution time. Components with frequent or large intermediate exchanges are generally favored for HPC execution, where direct memory and low-latency interconnects minimize communication cost.

Formally, when a component $v$ consumes output from $v'$, and their assigned locations differ, the transfer time is:

$$t_v^{xfer} = \frac{d_{v' \to v}}{BW_{\text{src}\to\text{dst}}}, \tag{4}$$

$$\text{cost}_{v' \to v}^{xfer} = c_{\text{xfer}}(d_{v' \to v}, \text{src}, \text{dst}) \tag{5}$$

where $d_{v' \to v}$ is the data volume to be transferred, $BW_{\text{src}\to\text{dst}}$ is the average available bandwidth between environments, and $c_{\text{xfer}}(\cdot)$ captures associated ingress and egress charges. All transfer costs are accumulated into the overall cost budget, ensuring that they are respected during scheduling. When components in a phase are co-located in the same serverless region, intermediate data is retained in the local shared file system, avoiding repeated transfers. If the next consumer component is on HPC, GridGreen pulls the data back to the HPC file system post-execution, again accounting for transfer time and reverse egress charges.

To reduce end-to-end latency, GridGreen pipelines transfers with computation where possible. For example, data generated early in a phase can begin transfer before the entire phase completes. This overlap allows dependent components in the next phase to start as soon as their inputs arrive. These data management strategies are tightly coupled with GridGreen's scheduling decisions. Components with high communication needs are preferentially scheduled on HPC, while serverless regions are used for compute-heavy, loosely coupled components where data exchange overhead remains modest. This data movement model enables GridGreen to balance the benefits of geographic carbon intensity variations with the practical constraints of serverless I/O, while staying within a strict cost budget and minimizing total service time.

**Algorithm 1:** GridGreen Scheduling Overview

```
   Input: DAG $G = (V, E)$, cost budget $B$, node range
          $\{N_{\min}, \ldots, N_{\max}\}$
   Output: Node count $N^*$, schedule $S^*$
1  foreach $N$ in $\{N_{\min}, \ldots, N_{\max}\}$ do
2      Predict $Q(N)$ from HPC job logs;
3  end
4  Forecast $CI_{hpc}(t), \{CI_r(t)\}$ using WattTime model;
5  Select $N^*$ minimizing queue–carbon–cost tradeoff;
6  foreach valid schedule $S$ over $G$ given $N^*$ do
7      Compute $T_{\text{total}} = Q(N^*) + \sum_p \max_{v \in \mathcal{V}_p}(t_v^{exec} + t_v^{xfer})$;
8      Compute $C_{\text{total}} = \sum_p \sum_{v \in \mathcal{V}_p}(E_v \cdot CI_v(t_v) + C_v^{emb})$;
9      Discard $S$ if $T_{\text{total}} > B$;
10 end
11 Select $S^*$ minimizing $T_{\text{total}} + \lambda C_{\text{total}}$, $\lambda$ is scaling factor;
12 Pre-warm serverless regions for offloaded components;
13 Adapt $S^*$ if $Q(N)$ or $CI_r(t)$ change significantly;
```

**Fallback due to Dynamic Queue and CI Changes.** GridGreen continuously monitors real-time HPC queue conditions and serverless region carbon intensity. If there is a sudden increase in queue load at HPC facility – such as arrival of higher-priority jobs, GridGreen reduces the number of allocated HPC nodes for future phases. Similarly, if a serverless region's carbon intensity rises significantly above forecasted values, components originally scheduled there are reassigned either to HPC facility (with co-location if needed) or to an alternate serverless region. In both cases, reassignment follows the same optimization strategy used in initial scheduling, selecting placements that minimize total service time and carbon footprint within the cost budget. Only components that have not yet begun execution are migrated; running components are left undisturbed. This adaptive fallback ensures GridGreen remains responsive to real-time conditions without violating execution constraints. Fig. 6 shows the interactions between design components of GridGreen.

**Exhaustive Search Complexity.** Our exhaustive search differs fundamentally from traditional brute-force approaches. GridGreen performs a simulation-based search over possible component assignments using pre-profiled execution times and energy measurements obtained offline for each HPC node and serverless region. This search evaluates assignments via lightweight simulation, not actual execution. Each evaluation involves only: (1) lookup of pre-profiled metrics (O(1)), (2) arithmetic for carbon calculations, and (3) maximum computation for phase completion time. The search scales through phase-wise decomposition (evaluating each phase independently), cost-based pruning (eliminating assignments exceeding budget constraints), and carbon threshold pruning (discarding assignments with excessive carbon footprint regions).

## 3.3 Implementation and Integration

To implement GridGreen, we combine the functionality of each design component in a unified software stack that runs on top of the home HPC facility's scheduler and leverages serverless providers via their respective command-line interfaces (e.g., AWS CLI). Upon receiving a workflow description, GridGreen parses the DAG structure and applies the node allocation logic first, calling a regression-based predictor to estimate queue wait time ($Q(N)$) for each possible node count ($N$). It then queries the WattTime-based carbon intensity forecasts [1] for the HPC facility location and each candidate serverless location.

After choosing $N$, GridGreen performs an offline brute-force search over the DAG to determine optimal component placement. This search operates on each phase of the DAG sequentially. For a phase $p$ with $n$ components, GridGreen enumerates all $k^n$ possible assignments ($k = 1$ HPC location + $k - 1$ serverless regions). For each assignment, GridGreen simulates execution by: (1) calculating each component's execution time on its assigned location using pre-profiled data; (2) adding transfer delays if input data comes from a different location; (3) computing the phase completion time as the maximum service time across all components in a phase; and (4) calculating the carbon footprint using energy profiles and predicted carbon intensity. We select the assignment minimizing a weighted combination of service time and carbon footprint, subject to the cost budget $B$. This search has time complexity $O(P \times k^n \times n)$ for $P$ phases and space complexity $O(n)$. For typical workflows with $n < 100$ components per phase and $k \leq 10$ locations, the search remains tractable, completing in under one minute for workflows with hours-long execution times.

The regression model predicting queue wait time $Q(N)$ for $N$ nodes is trained offline using the past month's HPC job logs. It employs a linear regression approach with features such as nodes requested, time of day, and day of the week, achieving an $R^2$ accuracy of over 0.8, consistent with prior HPC-scheduling work [41, 42]. To capture seasonal patterns and maintain accuracy, the model is retrained monthly. The regression has $O(m \log m)$ training complexity and is critical for determining the initial HPC allocation. Together with the brute-force search, which optimizes fine-grained component placement during execution, these components form the core of GridGreen's scheduling framework.

Once the plan is finalized, GridGreen provisions $N$ HPC nodes, stages HPC-based components there, and simultaneously registers serverless functions and their required data bundles for offloaded components. Each offloaded component is zipped along with its dependencies and input data, then uploaded as a deployable unit to the target serverless region. At runtime, whenever a new phase begins, GridGreen monitors queue load and carbon forecasts, triggers speculative pre-warming if necessary, and orchestrates the HPC-to-serverless data transfers required by the chosen schedule.

To handle the fallback mechanism in real time, the implementation continuously samples the HPC queue status and serverless carbon feeds. If queue delays inflate or carbon intensity rises unexpectedly in a certain region, GridGreen modifies assignments for the upcoming, not-yet-started components—subject to the same budget constraints and scheduling model used initially. Data management is transparently coordinated through remote elastic network file systems (e.g., AWS EFS), ensuring component inputs are staged properly. Reassignments affect only future components, and GridGreen maintains correctness while adapting to dynamic conditions. We summarize GridGreen's scheduling algorithm in Algorithm 1. GridGreen's implementation is based on Python3.0 and follows a methodology similar to the state-of-the-art carbon-unaware serverless-HPC scheduler MASHUP [55], which we use as a competing solution, with extensions including WattTime API integration for carbon intensity forecasting and RAPL for energy measurement. Serverless functions were deployed across multiple AWS regions to evaluate cross-region scheduling decisions and carbon-aware execution, as detailed in the next section.

## 4 Experimental Methodology

**Evaluated Workflows.** We evaluate GridGreen using three well-established bioinformatics workflows: *1000Genome*, *SRAsearch*, and *Epigenomics*. These are widely used in workflow systems and HPC scheduling research [33, 41, 68, 69, 76, 77], and feature diverse DAG patterns including fan-in, fan-out, and full inter-phase connectivity. Their varied communication and computation characteristics make them suitable for evaluating performance and sustainability-aware scheduling [55]. 1000Genome spans 2,506 components across 5 stages and processes 600 GB of genetic variation data. SRAsearch consists of 404 components and operates on 6 TB of biological sequence data using search-optimized compute kernels. Epigenomics performs DNA methylation analysis using 2,007 components across 9 stages, processing 5 TB of input. Together, these workflows capture real-world scientific workloads that stress compute, memory, and I/O subsystems [13, 45, 60]. Next, we discuss the competing solutions evaluated to determine GridGreen's effectiveness.

**Competing Solutions.** We compare GridGreen with different optimal solutions, which take into consideration only one GridGreen's optimization goals, and with the state-of-the-art serverless-HPC scheduler, which is unaware of carbon footprint.

*Carbon Optimal (Carbon-Opt).* This baseline uses the same node allocation strategy as GridGreen to determine the number of nodes in the home HPC facility. However, during execution, it places each component – either on the HPC facility or in one of the serverless regions, based solely on minimizing carbon footprint. At the start of each phase, it selects the placement that yields the lowest sum of operational and embodied emissions, using predicted carbon intensity and profiled energy usage. It does not consider queue wait time, execution time, or cost.

*Service-Time Optimal (Service-Time-Opt).* This baseline optimizes purely for total service time, including queue wait time, execution time, and data transfer overheads. The number of nodes allocated at the home HPC facility is chosen to minimize total expected service time based on queue predictions. During execution, each component is placed – either on the HPC facility or a serverless region, according to the fastest expected completion path. Carbon footprint and cost are not considered.

*Carbon-unaware Hybrid Serverles Scheduler, MASHUP [55].* It is the most relevant hybrid execution framework that profiles each component on serverless and local on-premise VM-based cluster environments to minimize service time and cost. It does not account for carbon footprint or HPC-specific factors such as queue wait time. Moreover, it operates in a single serverless region and does not support multi-region placement. In our evaluation, we implement Mashup's component-wise placement policy using their profiling methodology and assign all serverless components to the region with the lowest average CI to make the comparison representative.

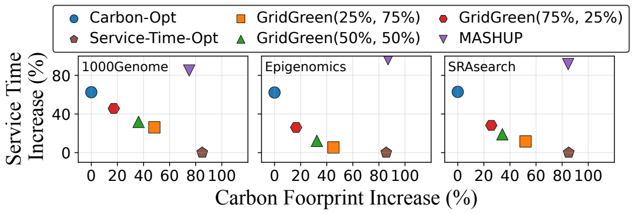


**Figure 7: GridGreen jointly optimizes the carbon footprint and service time of HPC workflows** (carbon footprint expressed as %-increase w.r.t Carbon-Opt and service time expressed as %-increase w.r.t Service-Time-Opt)**.**

**Accounting for the Embodied and Operational Carbon Footprint.** GridGreen uses widely-used public dataset [21, 43] to determine the embodied carbon of various hardware, which is based on the embodied carbon modeling tool ACT [27] and data from multiple sources, including Dell's carbon footprint data [7, 14, 64]. For accounting for the embodied carbon footprint of component execution via serverless functions, GridGreen accounts for the embodied carbon of the resources used during function execution and pre-warming, assuming unused resources may be shared across other cloud services. This is a standard methodology as proposed for serverless carbon accounting [54]. GridGreen adopts a standard four-year server lifetime model [24, 61], and amortizes the embodied carbon over this period based on the fraction of time a component uses computing resources. For nodes allocated at home HPC facility (dedicated nodes not shared), GridGreen accounts for their full embodied carbon during the allocation period.

GridGreen measures energy usage for HPC execution using RAPL [36] on bare-metal nodes and attributes the full node's energy, as nodes are exclusively allocated. For serverless functions, operational energy is estimated per function based on resource usage, using the methodology for serverless carbon footprint accounting as discussed in [54], similar to how embodied carbon is accounted for. The region-wise data on carbon intensity variation with time is gathered from WattTime [1], and GridGreen uses marginal carbon intensity, which reflects the emissions impact of adding additional load to the grid – making it more appropriate for scheduling decisions than average intensity [66, 73].

**Experimental Platform.** For the HPC facility, we use Intel Xeon Platinum 8175M nodes with 128 GB memory, and 10 Gbps network bandwidth, matching AWS m5 general-purpose instances. In our evaluation, we assume these HPC nodes are located in regions corresponding to major national laboratories. For the main set of experiments, we assume placement at Argonne National Laboratory and associate it with AWS's `us-east-2` region, the closest available AWS region. We use the marginal carbon intensity of this region, obtained from WattTime, as a proxy for Argonne's grid footprint. We also evaluate two additional assumed HPC deployments: Lawrence Berkeley National Lab (mapped to `us-west-1`) and Oak Ridge National Lab (mapped to `us-east-1`), and use the corresponding regional carbon intensities in each case.

For serverless execution, we deploy AWS Lambda functions in six regions: `us-west-1` (California), `us-east-1` (Virginia), `eu-west-1` (Ireland), `eu-central-1` (Germany), `ap-northeast-2` (Korea), and `sa-east-1` (Brazil). We use marginal carbon intensity data for each region during our evaluation period in March 2025. As

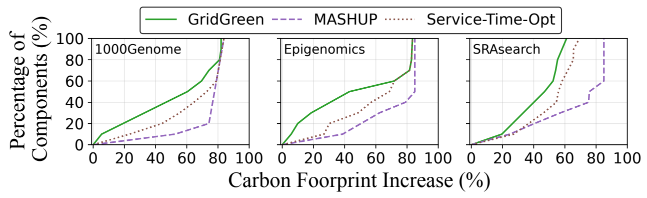


**Figure 8: GridGreen consistently improves carbon footprint across different components** (%-increase w.r.t Carbon-Opt)**.**

direct energy measurement is not supported on serverless, we follow serverless carbon accounting methodology from [54] to estimate per-function energy using profiling on equivalent bare-metal servers and compute operational carbon accordingly. Transfer-related emissions and delays are accounted for during multi-region placement of components in AWS. Execution cost is determined by standard rates charged by AWS Lambda. Inter-component communication in serverless is implemented using AWS Elastic File System (EFS), with coordination handled via `aws-cli` and `boto3`.

GridGreen's optimization is architecture-agnostic and applies equally to GPU-based workflows. It optimizes component placement based on execution profiles, CI, and queue times, which are metrics independent of compute architecture. GPU components would similarly benefit from carbon-aware scheduling because the framework's core mechanisms – profiling energy consumption, tracking regional CI variations, and managing queue delays – function identically across hardware types. The scheduling algorithm treats GPUs as higher-power compute resources, automatically accounting for their increased energy usage through the profiling phase while still leveraging the same carbon-aware placement strategies. For example, running 1000Genome GPU version on AWS g4 instances achieves carbon footprint and service time improvement within only 5-9% difference from CPU-based results.

**Evaluation Metrics.** We evaluate GridGreen using service time, and carbon footprint. Service time is defined as the sum of the HPC queue wait time and the total phase execution time, with each phase duration determined by the longest-running component. This also includes the data transfer overheads, warm start times (for kept-alive resources), and cold start delays incurred during workflow execution. Carbon footprint includes both operational and embodied emissions, including the carbon of the keep-alive period, along with carbon during execution. GridGreen is designed to operate under a cost constraint as a budget where the cost refers to the actual monetary charges incurred from the cloud provider, including function execution, data transfer, and storage. In most cases, carbon footprint results are reported as a percentage increase relative to the Carbon-Opt baseline, and service time as a percentage increase relative to the Service Time-Opt baseline.

## 5 Evaluation

In this section, we first evaluate the effectiveness of GridGreen and then understand the reasons behind its effectiveness. Thereafter, we test the robustness of GridGreen across diverse environments.

### 5.1 Effectiveness of GridGreen

**Comparison with Competing Solutions.** Fig. 7 compares GridGreen against competing strategies across the 1000Genome, Epigenomics, and SRAsearch workflows, showing the trade-offs between carbon footprint (expressed as %-increase w.r.t. Carbon-Opt) and service time (expressed as %-increase w.r.t Service-Time-Opt). Carbon-Opt achieves minimal carbon emissions but at the expense of higher service time (up to 80% for 1000Genome). Conversely, Service-Time-Opt minimizes service time but increases carbon emissions (up to 90% for 1000Genome). MASHUP performs poorly in both metrics. Its sub-optimal performance in terms of carbon footprint stems from fundamentally carbon-unaware scheduling. Even with our added modification of placing all components across all phases in the region with the lowest average carbon intensity (for a fair comparison), MASHUP does not account for temporal variation in carbon intensity across phases. In contrast, GridGreen adapts component placement at each phase based on real-time regional CI, avoiding this limitation. Additionally, MASHUP determines whether a component should run on serverless or on-premise cluster (HPC facility in this case) solely based on execution time. This may lead to choices of execution of components with high energy usage (and hence higher operational carbon) in a region with higher CI or long execution times (increasing embodied carbon amortization) in the server with higher embodied carbon (among on-premise HPC and serverless). MASHUP also ignores queue wait time, lacks speculative pre-warming, and does not coordinate I/O among serverless and the HPC facility, increasing service time. It also lacks fallback mechanisms for dynamic queue and CI shifts.

In contrast, GridGreen consistently achieves balanced outcomes across the workflows. Specifically, the default GridGreen (50,50), which equally weights carbon and service time optimization, achieves approximately 35% carbon footprint increase and 30% service time increase for 1000Genome. Adjusting the weights in GridGreen enables prioritization between carbon footprint and service time. The (25,75) configuration assigns 25% weight to carbon and 75% to service time, leading to lower service time at the cost of higher carbon. Conversely, (75,25) gives more importance to carbon reduction, resulting in lower emissions but higher service time. The (50,50) configuration balances both objectives equally and is referred to as the default GridGreen throughout this paper.

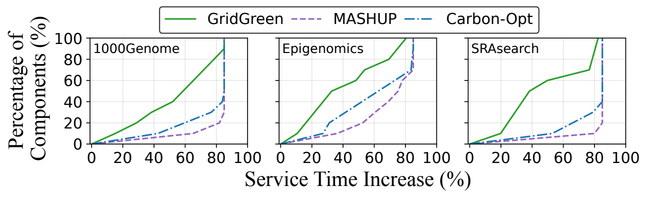


**Figure 9: GridGreen consistently improves service time across different components** (%-increase w.r.t Service-Time-Opt)**.**

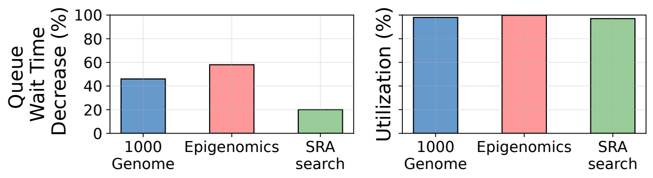


**Figure 10: GridGreen reduces queue wait time and improves HPC resource utilization** (queue wait time expressed as %-decrease w.r.t workflow execution completely in the HPC facility, without any serverless involvement)**.**

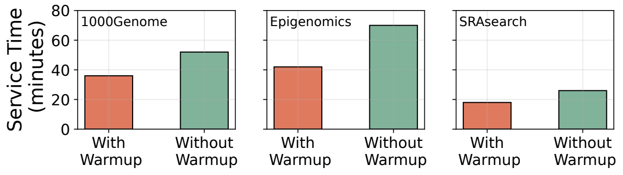


**Figure 11: Speculative warm-up consistently reduces the service time across all three evaluated workflows.**

Fig. 8 shows the cumulative distribution of carbon footprint increases across workflow components. GridGreen consistently provides a lower carbon footprint increase (w.r.t Carbon-Opt) compared to MASHUP and Service Time-Opt. For instance, around 50% of 1000Genome components in GridGreen incur a 50% or lower carbon increase, whereas MASHUP sees rapid escalation, reaching nearly 78% carbon increase for same proportion. Similarly, Fig. 9 demonstrates that across all components, GridGreen efficiently manages service time compared to MASHUP and Carbon-Opt. GridGreen's superior performance is due to adaptive scheduling considering the impact of the execution time of components in serverless region v/s HPC facility, speculative pre-warming strategy, and effective data coordination that significantly reduces I/O and cold start delays.

**Improving HPC Facility's Queue Wait Time and Utilization.** Recall from Sec. 2 that one key benefit of introducing serverless into HPC workflows is the potential to reduce queue wait time and improve overall system utilization. Fig. 10 confirms this: GridGreen reduces queue wait time by offloading components to serverless, with reductions of 47% for 1000Genome, 59% for Epigenomics, and 21% for SRAsearch. These savings result from requiring fewer HPC nodes, making smaller jobs easier to schedule. Simultaneously, GridGreen maintains high utilization of reserved HPC nodes – over 94% on average, since by design, only a limited number of nodes are allocated, and GridGreen selectively assigns components best suited for local execution based on their service time and carbon footprint. Next, we discuss the design components of GridGreen that results in its effectiveness in reducing carbon footprint and service time.

## 5.2 Reasons Behind GridGreen's Effectiveness

GridGreen allocates HPC nodes and serverless functions at a component level by solving an optimization problem that jointly minimizes service time and carbon footprint. Here, we analyze how these components contribute to GridGreen's overall effectiveness.

**Impact of Speculative Pre-Warming.** One of the key design components of GridGreen is its speculative pre-warming mechanism, which proactively initializes serverless functions before their actual invocation. This avoids cold start delays by ensuring that the function's code, data, and dependencies are already loaded in memory when execution begins. Fig. 11 shows the impact of this strategy: for 1000Genome, speculative pre-warming reduces average service time from 52 to 36 minutes; for Epigenomics, from 70 to 42 minutes. SRASearch also benefits significantly. Across all workflows, speculative pre-warming consistently lowers service time by mitigating initialization overhead, especially for components with large memory and data footprints.

**Impact of Data and I/O Management.** GridGreen incorporates explicit modeling and coordination of data transfers across HPC and serverless components, including intra- and inter-region communication. This design ensures that components with large or latency-sensitive data dependencies are either co-located or scheduled to minimize I/O overheads. Fig. 12 shows the performance benefit: for 1000Genome, service time drops from 52 to 21 minutes when data and I/O management is enabled. Epigenomics sees a reduction from 72 to 60 minutes, and SRASearch from 26 to 11 minutes. These improvements stem from GridGreen's ability to avoid redundant transfers, pipeline data movement, and handle serverless-to-serverless I/O via shared file systems like AWS EFS, while reserving tightly coupled components for HPC execution where direct communication is possible.

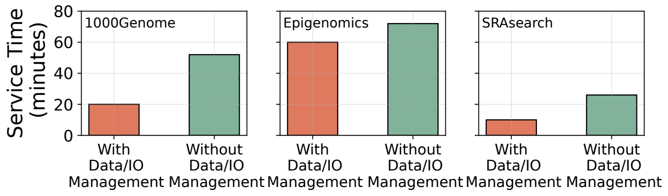


**Figure 12: GridGreen's data management helps to reduce I/O overhead of serverless computing.**

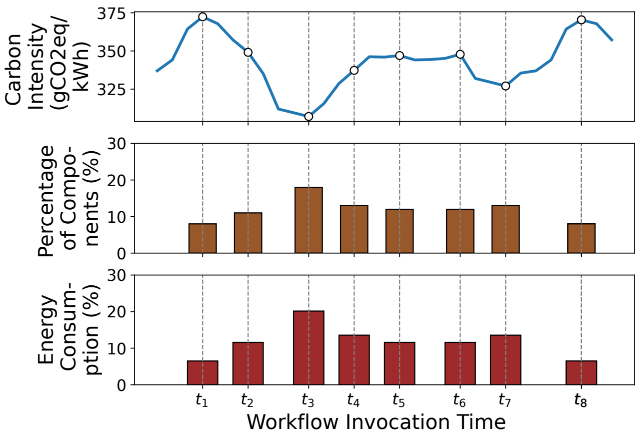


**Figure 13: Temporal analysis of carbon intensity (at home HPC facility – Argonne, at different periods of July 2024), component allocation by GridGreen, and energy consumption across workflow invocation times (**percentage of components and energy consumption at home HPC facility, Argonne, for executing 1000Genome at different periods in July 2024**).**

**Impact of Fallback and Component-Level Scheduling Optimization.** GridGreen performs fine-grained, component-level scheduling by solving an optimization that jointly considers energy consumption and carbon intensity at execution time. Combined with its fallback mechanism, GridGreen dynamically adapts to real-time variations in carbon intensity. As shown in Fig. 13, carbon intensity (at the home HPC facility, Argonne) varies significantly across submission times (top subplot), and GridGreen correspondingly adjusts the percentage of components allocated to the HPC facility (middle subplot) – favoring higher allocation when CI is low (e.g., around $t_3$) and reducing it when CI is high (e.g., $t_2$, $t_8$). This results in reduced energy usage at home HPC facility during high-CI periods (bottom subplot), as seen by the drop in HPC energy consumption at those times. This coordinated response is driven

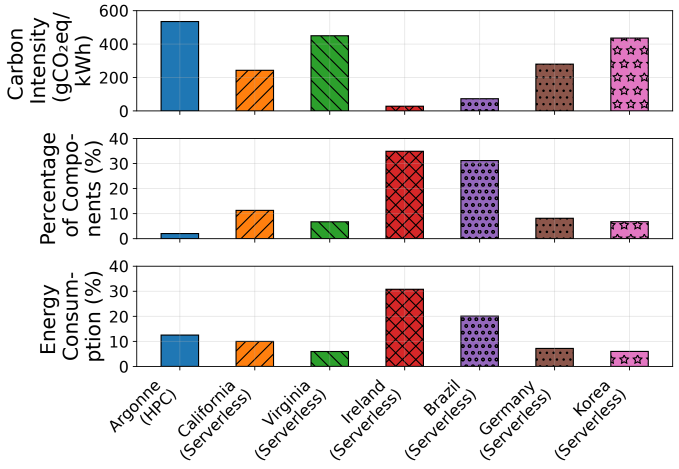


**Figure 14: Carbon intensity, component allocation, and energy consumption across HPC and serverless regions (executing 1000Genome on July 2024). GridGreen adaptively favors low-CI regions to minimize operational carbon footprint while meeting performance constraints.**

by GridGreen's scheduling optimization to minimize operational carbon (along with embodied) by aligning energy usage with favorable CI windows, and its fallback strategy, which shifts upcoming components to alternate regions when CI unexpectedly rises.

To further illustrate GridGreen's adaptive behavior, Fig. 14 presents the average carbon intensity across different regions (home HPC facility at Argonne and serverless regions) in July 2024, alongside the corresponding component allocations and energy consumption. We execute 1000Genome every day in July 2024 and show the average component allocations across all runs. The top subplot highlights large CI differences – ranging from 30 $gCO_2eq/kWh$ in Ireland to 540 in Illinois (Argonne – home HPC facility). In response, GridGreen adaptively allocates more components to regions with lower CI, as seen in the middle subplot, where Ireland and Brazil together receive over 65% of components, while high-CI regions like Korea and Argonne are less used. This behavior translates to the energy distribution (bottom subplot), with energy consumption skewed toward cleaner regions. These results reinforce that GridGreen's scheduler not only reacts to temporal CI variation (Fig. 13) but also exploits spatial CI differences across regions. This adaptive spatial placement is critical to carbon footprint minimization while still lowering service time and operating under cost constraints.

## 5.3 Robustness of GridGreen

**Performance with Different Cost Constraints.** We evaluate GridGreen's performance under varying budget constraints, where the cost expenditure is expressed as a percentage of the total cost that would be incurred if the entire workflow were executed using AWS serverless infrastructure. Fig. 15 presents GridGreen's relative performance (compared to Service-Time-Opt and Carbon-Opt) as the constraint is relaxed from 20% to 60%. With more budget flexibility, single-objective baselines like Service-Time-Opt and Carbon-Opt benefit more directly—allowing them to further reduce their targeted metric. As a result, GridGreen's relative gap appears to widen. However, in percentage terms, GridGreen's performance improves, achieving up to 20% reductions in both service time and carbon footprint with relaxed cost constraints.

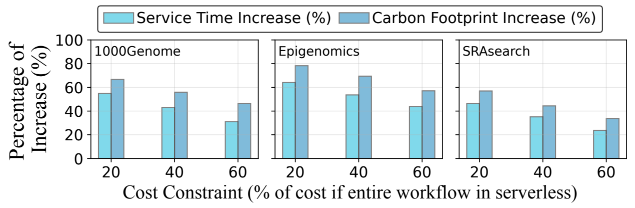


**Figure 15: GridGreen performs effectively under different cost constraints** (carbon footprint expressed as %-increase w.r.t Carbon-Opt and service time as %-increase w.r.t Service-Time-Opt, cost constraint is expressed as %-increase over serverless execution)**.**

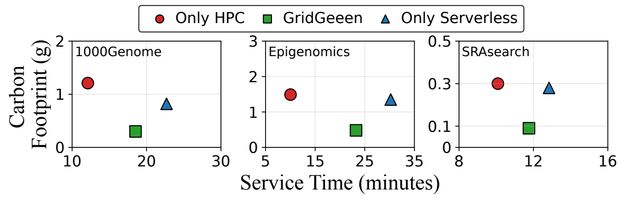


**Figure 16: Hybrid HPC-serverless execution balances GridGreen's in terms of carbon footprint and service time.**

**Performance across Different Deployment Modes.** To assess GridGreen's robustness across deployment modes, we compare GridGreen against two baselines: executing entirely in the home HPC center (Argonne) and fully in serverless regions, as defined in Sec. 4. As shown in Fig. 16, GridGreen consistently achieves a better balance between service time and carbon footprint. While HPC-only yields the fastest service time, it suffers from high carbon emissions due to operating in a fixed, high-CI region. Serverless-only reduces carbon emissions but incurs longer service times due to cold starts and I/O overheads. By selectively combining both environments, GridGreen outperforms each individual strategy in at least one metric and achieves competitive overall performance. This balanced performance results from GridGreen's ability to place compute-intensive components with low data dependencies on serverless (exploiting regional CI variations) while retaining tightly-coupled, communication-heavy components on HPC nodes. GridGreen's hybrid approach particularly excels for workflows with heterogeneous component characteristics, where neither pure strategy can optimize both metrics effectively.

**Performance at Different Home HPC Facilities.** We evaluate GridGreen's performance by selecting different home HPC facilities: Argonne (Illinois), Lawrence Berkeley (California), and Oak Ridge (Tennessee). As shown in Fig. 17, GridGreen consistently achieves a balanced trade-off between service time and carbon footprint across all three sites. The effectiveness varies based on each facility's regional grid characteristics and carbon intensity profiles. At Argonne, while Carbon-Opt suffers approximately 60% service time increase and Service-Time-Opt incurs 60% carbon footprint increase, GridGreen maintains a balanced 40% service time increase with only 25% carbon footprint increase. Berkeley's cleaner grid enables even better performance – GridGreen achieves merely 15% carbon footprint increase while limiting service time increase to 35%,

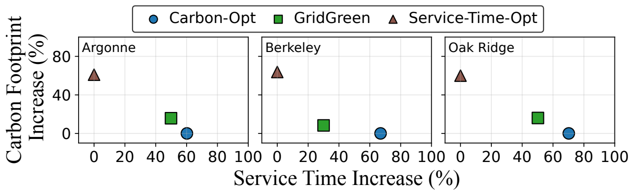


**Figure 17: GridGreen's effectiveness is consistent across HPC facilities** (carbon footprint expressed as %-increase w.r.t Carbon-Opt and service time as %-increase w.r.t Service-Time-Opt).

compared to Carbon-Opt's 70% service time penalty. Oak Ridge exhibits similar balanced characteristics. This consistent performance across geographically diverse facilities demonstrates GridGreen's robustness to varying regional carbon intensities and queue behaviors. The framework achieves this by dynamically adjusting component allocation based on real-time carbon intensity – allocating more components locally when Berkeley's CI is low (due to solar/wind availability) while offloading up to 65% of components to cleaner serverless regions when Argonne's grid exhibits high CI periods. These results validate GridGreen's practical applicability across different HPC deployment scenarios without requiring facility-specific tuning or resource allocations.

## 6 Discussion on Future Directions

**Integration with production HPC schedulers.** While the current implementation of GridGreen operates as a standalone framework, practical deployment requires integration with established HPC schedulers like SLURM and PBS [50, 75]. Future work could implement GridGreen as a scheduler plugin that intercepts job submissions, performs carbon-aware planning, and translates decisions into native scheduler commands. Specifically, GridGreen could maintain a daemon process that monitors the scheduler's job queue, identifies workflow DAGs through job dependencies or metadata tags, and dynamically adjusts resource requests before submission. The serverless offloading logic could be embedded as custom job prologue/epilogue scripts that detect components and redirect their execution to cloud. This would preserve existing user interfaces and administrative controls while enabling carbon-aware scheduling without modifying core scheduler code.

**Multi-tenant carbon fairness.** Production HPC facilities serve multiple users whose workflows compete for resources and carbon reduction opportunities. Future work can develop fairness mechanisms that prevent carbon-efficient scheduling from being monopolized by specific users or workflows. This could involve carbon credit systems where users accumulate credits for accepting longer queue times or performance degradation in favor of cleaner execution windows. Alternatively, facilities could implement carbon budgets alongside traditional allocation metrics, ensuring each user receives proportional access to low-carbon time slots [29, 34]. Such mechanisms would require policy frameworks that balance individual optimization with system-wide sustainability goals.

**Incorporating additional environmental metrics.** While GridGreen focuses on carbon emissions, it can be extended to address other sustainability factors that also exhibit spatio-temporal variation. Water usage for datacenter cooling varies significantly by region [26] – facilities in arid areas rely on energy-intensive air cooling while others consume substantial water resources, creating different environmental trade-offs. Similarly, fluorinated compounds used in equipment manufacturing contribute significantly to embodied emissions and face varying regional regulations [9]. Future work could enhance GridGreen's multi-objective framework to incorporate these complementary metrics to enable component placement and HPC datacenter hardware procurement decisions.

## 7 Related Work

**Serverless for HPC Workflows.** Many previous research works have focused on designing serverless solutions for HPC workflows. DayDream [56] and StarShip [10] are serverless workflow schedulers, but both operate entirely in serverless environments, unlike GridGreen, which is a hybrid HPC-serverless framework. DayDream focuses on mitigating cold starts, while StarShip reduces I/O overhead. Neither system considers carbon footprint in scheduling decisions. Thurimella *et al.* [67] explores the feasibility and benefits of utilizing serverless computing paradigms for HPC workloads that exhibit dynamic behavior. SwitchFlow [17] proposes a mechanism to intelligently decompose and execute different stages of a workflow across various serverless platforms, selecting the most suitable framework for each task based on its specific requirements. Copik *et al.* [19] investigates the concept of software resource disaggregation in HPC using serverless computing. But none of these works tries to reduce the carbon footprint of HPC workflow execution.

**Carbon-Aware HPC Systems.** Recent work has explored carbon-aware computing in HPC and cloud systems. Li *et al.* [38] quantify the environmental impact of HPC and propose footprint estimation methods. Wassermann *et al.* [72] attribute HPC carbon emissions to users based on resource usage. EcoFreq [37] reduces environmental impact by tuning power use based on real-time grid conditions. Clover [39] optimizes ML inference deployment with carbon awareness. EcoLife [32] targets carbon-aware scheduling for general serverless workloads. In contrast, GridGreen focuses on compute-intensive HPC workflows and jointly addresses sustainability (carbon footprint) and performance (service time of workflow components) in hybrid serverless-HPC environments.

## 8 Conclusion

GridGreen is the first system to integrate serverless computing in traditional HPC facilities for executing workflows to minimize carbon footprint and service time, under cost constraints. By leveraging the flexibility of serverless, GridGreen dynamically adapts to spatio-temporal variations in carbon intensity to improve sustainability. It incorporates component-level optimization, speculative pre-warming, I/O-aware data management, and fallback adaptation to jointly minimize carbon footprint and service time under user-defined cost constraints. Evaluations on real scientific workflows across diverse leadership-scale facility locations show that GridGreen consistently performs well to meet its objectives.

**Acknowledgments:** We thank the anonymous reviewers for their valuable feedback. This work is supported by the University of Utah's Kahlert School of Computing, John and Marcia Price College of Engineering, and Scientific Computing & Imaging Institute.

## References

[1] 2025. Home - WattTime — watttime.org. https://watttime.org/.

[2] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon explorer: A holistic framework for designing carbon aware datacenters. In *ACM ASPLOS*. 118–132.

[3] Amazon. 2021. *Amazon Sustainability Report*. https://sustainability.aboutamazon.com/2021-sustainability-report.pdf

[4] Anders SG Andrae and Tomas Edler. 2015. On global electricity usage of communication technology: trends to 2030. *Challenges* 6, 1 (2015), 117–157.

[5] Pau Andrio, Adam Hospital, Javier Conejero, Luis Jordá, Marc Del Pino, Laia Codo, Stian Soiland-Reyes, Carole Goble, Daniele Lezzi, Rosa M Badia, et al. 2019. BioExcel Building Blocks, a software library for interoperable biomolecular simulation workflows. *Scientific data* 6, 1 (2019), 1–8.

[6] Apple. 2022. *Environmental Progress Report*. https://www.apple.com/environment/pdf/Apple_Environmental_Progress_Report_2022.pdf

[7] Clément Auger, Benoit Hilloulin, Benjamin Boisserie, Maël Thomas, Quentin Guignard, and Emmanuel Rozière. 2021. Open-source carbon footprint estimator: Development and university declination. *Sustainability* 13, 8 (2021), 4315.

[8] Troy Baer, Paul Peltz, Junqi Yin, and Edmon Begoli. 2015. Integrating apache spark into pbs-based hpc environments. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. 1–7.

[9] Rohan Basu Roy, Raghavendra Kanakagiri, Yankai Jiang, and Devesh Tiwari. 2025. ForgetMeNot: Understanding and modeling the impact of forever chemicals toward sustainable large-scale computing. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 9, 2 (2025), 1–26.

[10] Rohan Basu Roy and Devesh Tiwari. 2024. Starship: Mitigating i/o bottlenecks in serverless computing for scientific workflows. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–29.

[11] André Bauer, Haochen Pan, Ryan Chard, Yadu Babuji, Josh Bryan, Devesh Tiwari, Ian Foster, and Kyle Chard. 2024. The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud. *Future Generation Computer Systems* 153 (2024), 558–574.

[12] Andrea R Beccari, Carlo Cavazzoni, Claudia Beato, and Gabriele Costantino. 2013. LiGen: a high performance workflow for chemistry driven de novo design.

[13] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. 2008. Characterization of scientific workflows. In *2008 third workshop on workflows in support of large-scale science*. IEEE, 1–10.

[14] Sarah B Boyd. 2011. *Life-cycle assessment of semiconductors*. Springer Science & Business Media.

[15] Zhiwei Cao, Xin Zhou, Han Hu, Zhi Wang, and Yonggang Wen. 2022. Towards a systematic survey for carbon neutral data centers. *IEEE Communications Surveys & Tutorials* (2022).

[16] Jichuan Chang, Justin Meza, Parthasarathy Ranganathan, Amip Shah, Rocky Shih, and Cullen Bash. 2012. Totally green: evaluating and designing servers for lifecycle environmental impact. *ACM SIGPLAN Notices* 47, 4 (2012), 25–36.

[17] Hao Chen, Yucong Dong, Xin Wen, and Zichen Xu. 2024. SwitchFlow: Optimizing HPC Workflow Performance with Heterogeneous Serverless Frameworks. In *2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS)*. 286–293.

[18] United States International Trade Commission. 2021. *Data Centers Around the World: A Quick Look*. https://www.usitc.gov/publications/332/executive_briefings/ebot_data_centers_around_the_world.pdf

[19] Marcin Copik, Marcin Chrapek, Larissa Schmid, Alexandru Calotoiu, and Torsten Hoefler. 2024. Software Resource Disaggregation for HPC with Serverless Computing. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 139–156.

[20] Marcin Copik, Konstantin Taranov, Alexandru Calotoiu, and Torsten Hoefler. 2023. rFaaS: Enabling High Performance Serverless with RDMA and Leases. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 897–907.

[21] Benjamin Davy. 2021. Building an AWS EC2 carbon emissions dataset. *Medium, September* (2021).

[22] W Freudling, M Romaniello, DM Bramich, P Ballester, V Forchi, CE García-Dabló, S Moehler, and MJ Neeser. 2013. Automated data reduction workflows for astronomy-The ESO Reflex environment. *Astronomy & Astrophysics* 559 (2013), A96.

[23] Google. 2022. *Google Environmental Report*. https://www.gstatic.com/gumdrop/sustainability/google-2022-environmental-report.pdf

[24] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. 2011. Benefits and limitations of tapping into stored energy for datacenters. In *Proceedings of the 38th annual international symposium on Computer architecture*. 341–352.

[25] Yi Gu and Chandu Budati. 2020. Energy-aware workflow scheduling and optimization in clouds using bat algorithm. *Future Generation Computer Systems* 113 (2020), 106–112.

[26] Pranjol Sen Gupta, Md Rajib Hossen, Pengfei Li, Shaolei Ren, and Mohammad A Islam. 2024. A dataset for research on water sustainability. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*. 442–446.

[27] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: Designing sustainable computer systems with an architectural carbon modeling tool. In *ACM/IEEE ISCA*. 784–799.

[28] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. 2022. Chasing carbon: The elusive environmental footprint of computing. *IEEE Micro* 42, 4 (2022), 37–47.

[29] Leo Han, Jash Kakadia, Benjamin C Lee, and Udit Gupta. 2025. Fair-CO2: Fair attribution for cloud carbon emissions. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 646–663.

[30] Walid A Hanafy, Roozbeh Bostandoost, Noman Bashir, David Irwin, Mohammad Hajiesmaili, and Prashant Shenoy. 2024. The war of the efficiencies: Understanding the tension between carbon and energy optimization. *ACM SIGENERGY Energy Informatics Review* 4, 3 (2024), 87–93.

[31] Walid A Hanafy, Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. 2023. CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency. *arXiv preprint arXiv:2302.08681* (2023).

[32] Yankai Jiang, Rohan Basu Roy, Baolin Li, and Devesh Tiwari. 2024. EcoLife: Carbon-Aware Serverless Function Scheduling for Sustainable Computing. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[33] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. 2009. Scientific workflow applications on Amazon EC2. In *2009 5th IEEE international conference on e-science workshops*. IEEE, 59–66.

[34] Alok Kamatar, Maxime Gonthier, Valerie Hayot-Sasson, Andre Bauer, Marcin Copik, Torsten Hoefler, Raul Castro Fernandez, Kyle Chard, and Ian Foster. 2025. Core Hours and Carbon Credits: Incentivizing Sustainability in HPC. *arXiv preprint arXiv:2501.09557* (2025).

[35] Daniel Kelly, Frank Glavin, and Enda Barrett. 2020. Serverless Computing: Behind the Scenes of Major Platforms. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 304–312.

[36] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 3, 2 (2018), 1–26.

[37] Oleksiy M Kozlov and Alexandros Stamatakis. 2024. EcoFreq: compute with cheaper, cleaner energy via carbon-aware power scaling. In *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*. Prometeus GmbH, 1–12.

[38] Baolin Li, Rohan Basu Roy, Daniel Wang, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Toward sustainable hpc: Carbon footprint estimation and environmental implications of hpc systems. In *ACM/IEEE SC*. 1–15.

[39] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Clover: Toward sustainable ai with carbon-aware machine learning inference service. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[40] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Green Carbon Footprint for Model Inference Serving via Exploiting Mixed-Quality Models and GPU Partitioning. *arXiv preprint arXiv:2304.09781* (2023).

[41] Weiling Li, Yunni Xia, Mengchu Zhou, Xiaoning Sun, and Qingsheng Zhu. 2018. Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. *IEEE Access* 6 (2018), 61488–61502.

[42] Zhengchun Liu, Ryan Lewis, Rajkumar Kettimuthu, Kevin Harms, Philip Carns, Nageswara Rao, Ian Foster, and Michael E Papka. 2020. Characterization and identification of HPC applications at leadership computing facility. In *Proceedings of the 34th ACM International Conference on Supercomputing*. 1–12.

[43] Romain Lorenzini. 2021. *Digital & environment: How to evaluate server manufacturing footprint, beyond greenhouse gas emissions?* https://www.boavizta.org/en/blog/empreinte-de-la-fabrication-d-un-serveur

[44] Diptyaroop Maji, Prashant Shenoy, and Ramesh K Sitaraman. 2022. CarbonCast: multi-day forecasting of grid carbon intensity. In *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 198–207.

[45] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research* 20, 9 (2010), 1297–1303.

[46] Paul Messina. 2017. The exascale computing project. *Computing in Science & Engineering* 19, 3 (2017), 63–67.

[47] Meta. 2021. *Meta Sustainability Report*. https://sustainability.fb.com/wp-content/uploads/2022/06/Meta-2021-Sustainability-Report.pdf

[48] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. 2019. Agile cold starts for scalable serverless. In *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. 1–15.

[49] Panayiotis Neophytou, Roxana Gheorghiu, Rebecca Hachey, Timothy Luciani, Di Bao, Alexandros Labrinidis, Elisabeta G Marai, and Panos K Chrysanthis. 2012.

Astroshelf: understanding the universe through scalable navigation of a galaxy of annotations. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 713–716.

[50] Bill Nitzberg, Jennifer M Schopf, and James Patton Jones. 2004. PBS Pro: Grid computing and scheduling attributes. In *Grid resource management: state of the art and future trends*. Springer, 183–190.

[51] Kary Ocaña, Silvia Benza, Daniel De Oliveira, Jonas Dias, and Marta Mattoso. 2014. Exploring large scale receptor-ligand pairs in molecular docking workflows in HPC clouds. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 536–545.

[52] Michael E Papka, Jini Ramprakash, Robert Scott, Doug Waldron, Adrian Pope, Aditya Tanikanti, Dana Silvestri, Jeffrey Neel, Laura Wolf, Avanthi Mantrala, et al. 2023. *2022 Operational Assessment Report-Argonne Leadership Computing Facility*. Technical Report. Argonne National Laboratory (ANL), Argonne, IL (United States).

[53] Elaina Present, Pieter Gagnon, Eric JH Wilson, Noel Merket, Philip R White, and Scott Horowitz. 2024. *Choosing the best carbon factor for the job: Exploring available carbon emissions factors and the impact of factor selection*. Technical Report. National Renewable Energy Laboratory (NREL), Golden, CO (United States).

[54] Rohan Basu Roy, Raghavendra Kanakagiri, Yankai Jiang, and Devesh Tiwari. 2024. The Hidden Carbon Footprint of Serverless Computing. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 570–579.

[55] Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, and Devesh Tiwari. 2022. Mashup: making serverless computing useful for HPC workflows via hybrid execution. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 46–60.

[56] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. DayDream: executing dynamic scientific workflows on serverless platforms with hot starts. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–18.

[57] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. IceBreaker: warming serverless functions better with heterogeneity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 753–767.

[58] Amit Samanta, Faraz Ahmed, Lianjie Cao, Ryan Stutsman, and Puneet Sharma. 2023. Persistent memory-aware scheduling for serverless workloads. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 615–621.

[59] Amit Samanta and Ryan Stutsman. 2024. Fair, Efficient Multi-Resource Scheduling for Stateless Serverless Functions with Anubis. In *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 106–112.

[60] Greg Schulz. 2011. *Cloud and virtual data storage networking*. CRC Press.

[61] Arman Shehabi, Sarah Smith, Dale Sartor, Richard Brown, Magnus Herrlin, Jonathan Koomey, Eric Masanet, Nathaniel Horner, Inês Azevedo, and William Lintner. 2016. United states data center energy usage report. (2016).

[62] Ana Luisa Veroneze Solórzano, Kento Sato, Keiji Yamamoto, Fumiyoshi Shoji, Jim M Brandt, Benjamin Schwaller, Sara Petra Walton, Jennifer Green, and Devesh Tiwari. 2024. Toward Sustainable HPC: In-Production Deployment of Incentive-Based Power Efficiency Mechanism on the Fugaku Supercomputer. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[63] Abel Souza, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. 2023. 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*. 252–265.

[64] Markus Stutz, Scott O'Connell, and John Pflueger. 2012. Carbon footprint of a dell rack server. In *2012 Electronics Goes Green 2012+*. IEEE, 1–5.

[65] Estela Suarez, Jorge Amaya, Martin Frank, Oliver Freyermuth, Maria Girone, Bartosz Kostrzewa, and Susanne Pfalzner. 2025. Energy Efficiency trends in HPC: what high-energy and astrophysicists need to know. *arXiv preprint arXiv:2503.17283* (2025).

[66] Thanathorn Sukprasert, Noman Bashir, Abel Souza, David Irwin, and Prashant Shenoy. 2024. On the Implications of Choosing Average versus Marginal Carbon Intensity Signals on Carbon-aware Optimizations. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*. 422–427.

[67] Vijay Thurimella, Philipp Raith, Rolando P. Hong Enriquez, Anderson Andrei Da Silva, Gourav Rattihalli, Ada Gavrilovska, and Dejan Milojicic. 2024. Serverless Computing for Dynamic HPC Workflows. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2096–2103.

[68] Amandeep Verma and Sakshi Kaushal. 2017. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Comput.* 62 (2017), 1–19.

[69] Laurens Versluis, Erwin Van Eyk, and Alexandru Iosup. 2018. An analysis of workflow formalisms for workflows with complex non-functional requirements. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. 107–112.

[70] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *2018 {USENIX} Annual Technical Conference ({USENIX} {ATC} 18)*. 133–146.

[71] Christian Wassermann, Mario Bielert, Gert Vanberg, Daniel Hackenberg, Christian Terboven, and Matthias S Müller. 2024. Calculating User-Centric Carbon Footprints for HPC. In *2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*. IEEE, 26–35.

[72] Christian Wassermann, Mario Bielert, Gert Vanberg, Daniel Hackenberg, Christian Terboven, and Matthias S. Müller. 2024. Calculating User-Centric Carbon Footprints for HPC. In *2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*. 26–35.

[73] WattTime. 2024. The methodology behind our latest global data expansion - WattTime — watttime.org. https://watttime.org/news-and-insights/the-methodology-behind-our-latest-global-data-expansion/.

[74] Bin Yang, Hao Wei, Wenhao Zhu, Yuhao Zhang, Weiguo Liu, and Wei Xue. 2024. Full lifecycle data analysis on a large-scale and leadership supercomputer: what can we learn from it?. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 917–933.

[75] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*. Springer, 44–60.

[76] Shuai Zeng, Zhen Lyu, Siva Ratna Kumari Narisetti, Dong Xu, and Trupti Joshi. 2018. Knowledge Base Commons (KBCommons) v1. 0: A multi OMICS'web-based data integration framework for biological discoveries. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 589–594.

[77] Zhaomeng Zhu, Gongxuan Zhang, Miqing Li, and Xiaohui Liu. 2015. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Transactions on parallel and distributed Systems* 27, 5 (2015), 1344–1357.