Sensitivity Analysis of Carbon-Aware Scheduling in Cloud Computing: A Simulation-Based Study

Sakshi Jamdade[1], Jyoti Raut[2], Manya Rathore[3], Aditi Kandari[4], Ms. Pranita Manjare[5]

Students, Department of MCA, D.Y. Patil Institute of Master of Computer Applications and Management, Akurdi, Pune.

5 Associate Professor, Department of MCA, D.Y. Patil Institute of Master of Computer Applications and Management, Akurdi, Pune.

Abstract

Data centres can trim their emissions by running flexible work while the grid is cleaner, but how large that trim appears depends heavily on the modelling choices behind the number. This study examines that dependence in a deliberately simple setting: ten synthetic jobs, 324 actual half-hourly Great Britain carbon-intensity readings across seven days (24–30 August 2026), a fixed 50 W per CPU core, and a scheduler given perfect foresight of the intensity that follows. With zero waiting allowed, estimated savings are nothing; grant up to 180 minutes and they reach about 4%. Those figures ignore each job's own deadline and the limited pool of cores. Introducing both with a mixed-integer formulation reduces the estimate to approximately 2.6%. Every value here is a deterministic best-case simulation rather than a measurement from a live data centre, so the short record, invented jobs, rough power assumption, and look-ahead signal all restrict how far the conclusions travel.

Keywords: carbon-aware scheduling; cloud computing; carbon intensity; workload shifting; sensitivity analysis; simulation

1. Introduction

Cloud services run on large server fleets whose electricity use carries a carbon cost. Whether a job is carbon-heavy depends on two things: how much power it consumes and how dirty the grid is at the moment it runs. Because grid carbon intensity falls and rises through the day, work that can be postponed to a cleaner period has a smaller estimated footprint even though the same computation is performed [1].

Carbon-aware scheduling exploits exactly this rhythm, choosing when flexible jobs should run. The benefit it delivers is far from constant — it depends on how long a job may wait, how long it takes, how many cores it needs, and the shape of the intensity curve. A policy that performs well under one set of conditions may perform poorly under another.

This paper uses simulation to trace that dependence. We vary the scheduling window, the measured day-by-day intensity, deliberately modified intensity profiles, the size of the workload, and the workload's configuration, keeping the energy model at a constant 50 W per core and intensity samples at 30-minute resolution. The goal is to identify which assumptions most strongly drive the estimated savings.

1.1 Statement of the Problem

Deferring flexible work to cleaner hours can cut emissions, but by how much is uncertain. A single number taken from one workload, one brief observation period, or one deferral limit can therefore give a misleading picture.

Our scope is deliberately tight. Within one transparent model, we ask how the deferral allowance and the timing of intensity shape estimated emissions, and how much confidence those estimates deserve given that the scheduler sees the future and ignores shared-hardware limits. We treat the study as an exploratory benchmark and make no claim about savings in production.

1.2 Objectives of the Research

The objectives of this study are:

To assess how the size of the scheduling-flexibility window changes estimated carbon savings.

To investigate how job runtime, core requirement, and permitted delay affect those savings.

To identify which modelling assumptions most influence the estimated benefit.

1.3 Research Question

RQ1: In this simplified model, how do the size of the scheduling window, the workload's configuration, and the temporal ordering of real grid carbon-intensity measurements change estimated operational CO_2 emissions relative to starting every job on arrival?

The study is descriptive and deterministic: we compare predefined scenarios instead of running inferential tests, so we claim neither population-level effects nor statistical significance.

2. Literature Review

Postponing flexible workloads to greener periods has repeatedly been associated with lower estimated operational emissions. Radovanović et al. (2023) reported this in live data-centre operations; Bashir et al. (2024) showed that the accounting method chosen alters the scheduling decision; and Asadov et al. (2025) surveyed spatial and temporal shifting along with its limitations [1–3].

Meaningful carbon figures also depend on transparent energy assumptions, emission factors, and temporal resolution. The Software Carbon Intensity specification from the Green Software Foundation offers accounting guidance [4], and the half-hourly grid values used in this study come from the UK Carbon Intensity API [5]. Wider context on energy-aware cloud scheduling appears in work by Kaur et al., Verma et al., and Biswas et al. [6–8].

More recent studies widen the scope. Deferral has been studied under different time budgets and forecast error [10] and applied to a live batch pipeline whose savings varied with season and time of arrival [20]. Instead of merely delaying jobs, carbon-scaling and carbon-aware provisioning adjust resource allocation to match grid intensity [11], [12]. Orchestration schedulers for Kubernetes and multi-cluster environments exploit spatial and temporal intensity differences while safeguarding service-level objectives [13], [14], [15], and one study frames day-ahead allocation across a network of data centres as a game [16]. Because accounting choices can substantially change reported savings, assumptions must remain transparent [9] — a point reinforced by work on forecasting and robustness under uncertainty [17], on carbon-aware workflow scheduling with fixed mappings and deadlines [18], and by field surveys [19].

Research Gap and Contribution: Prior work confirms both the value of carbon-aware scheduling and the influence of accounting decisions, but it rarely isolates how much the estimate moves when one assumption changes. This paper introduces no new algorithm; its contribution is a controlled sensitivity analysis together with a direct comparison of (i) an independent-job oracle that respects per-job delay limits but ignores shared capacity and (ii) a mixed-integer model that respects both the delay limits and a shared core budget. Any gap we measure belongs to this workload, capacity, and signal and does not generalise to all cloud workloads.

3. Research Methodology

3.1 Research Design

We adopt a quantitative, simulation-based design to trace how estimated carbon savings respond to changes in workload characteristics, available flexibility, and grid carbon intensity. In the baseline, every job starts the moment it arrives. The carbon-aware version evaluates all permitted start times within the window and selects the one that minimises estimated emissions over the entire run. No real platform, CPU contention, or inter-job interference is modelled, so the outputs are comparative estimates limited to the assumptions stated here.

The study comprises five experiment families: scheduling flexibility, day-to-day variation, alternative intensity profiles, workload size, and workload configuration. In each family one parameter is varied while the rest are held constant, which keeps the comparisons fair. Every reported saving is a simulation result and says nothing about a deployed system.

Table 1. Experimental design and parameters varied.

Experiment	Varied parameter	Values / scenarios	Workload and signal	Outcome

Flexibility	Global delay window	0, 30, 60, 120, 180 min	10 jobs; each available date	Aggregate gCO_2 and % saved
Daily variation	Experiment date	Seven dates; 180 min	10 jobs repeated per date	Daily and mean daily % saved
Intensity profile	Temporal profile	Actual; ×1.5; reversed	10 jobs; 60 min	gCO_2 and % saved
Workload size	Job subset	J1–J5 vs full workload	Small subset; 60 min	gCO_2 and % saved
Runtime	Runtime per job	15, 30, 60, 90, 120 min	Uniform change; 60 min	% saved
CPU cores	Cores per job	1, 2, 4, 8, 16	Uniform change; 60 min	% saved
Delay tolerance	Global delay	15, 30, 60, 120, 180 min	Uniform delay; 10 jobs	% saved

3.2 Data Collection

Grid data come from the actual-intensity field of the UK Carbon Intensity API for Great Britain [5]. A script polls the API each day, converts UTC timestamps to India Standard Time (IST), and keeps only records that contain an actual value. The resulting CSV contains 324 half-hourly rows from 24 August 2026 04:30 IST to 30 August 2026 22:00 IST, fewer than the 336 points a full week would give, with daily counts of 39, 48, 48, 48, 48, 48, and 45 for 24–30 August. Missing values are not interpolated. Day boundaries are defined using IST after conversion rather than Great Britain civil dates, and the series reflects the national grid, not any particular cloud region. Because launch times are selected using the actual intensity that follows, the scheduler effectively possesses foresight; the results are oracle-style estimates rather than what a forecast-based scheduler would achieve.

Ten synthetic jobs (J1–J10) run independently on each date, so the seven-date total sums daily runs rather than forming one continuous arrival stream. Arrival times are given in IST clock time. Each job has an arrival time, a runtime, a CPU-core requirement, and a max_delay_minutes value, and J1–J5 constitute the smaller subset. Permitted start times are the half-hourly points from arrival up to arrival plus the global window (0, 30, 60, 120, or 180 minutes), and the job is placed at the candidate that minimises estimated emissions over its full run. This scheduler does not enforce individual max_delay_minutes, models no CPU contention, and prevents no overlaps; the delay test therefore exercises a global window rather than per-job deadlines, and its output need not form a valid combined schedule.

Scheduling procedure

Algorithm 1. Retrospective independent-job scheduling.

1. For each available calendar date, select that date's available actual-intensity records and repeat the same workload with arrival times interpreted as IST.

2. For each job, calculate baseline emissions by starting at its arrival time and allocating energy over each overlapping 30-minute interval.

3. Enumerate available half-hourly candidate starts from arrival time through arrival time plus the global flexibility window.

4. For each candidate, calculate emissions over the full runtime using the observed actual-intensity series. If any required interval is unavailable, the implementation raises an error rather than imputing a value.

5. Select the candidate with the minimum calculated emissions. Ties follow the input row order; jobs are optimized independently and are not checked against shared CPU capacity or individual max_delay_minutes.

6. Sum job emissions to obtain daily baseline and carbon-aware totals; aggregate daily totals for the flexibility experiment and calculate savings as the baseline minus carbon-aware emissions.

Feasibility comparison implementation: a separate MILP gives each job a single half-hour-aligned start, respects that job's configured maximum delay, and limits the total CPU demand to the tested capacity in

every slot the job occupies. Every slot a job touches is counted in full against it — a deliberately conservative, slot-level simplification instead of continuous capacity tracking. Emissions are minimised under the same 50 W/core model and actual intensity values, and a companion solve minimises total delay over the same feasible set. These constraints apply only to this additional experiment and do not alter the original flexibility results.

MILP formulation

Let J be the set of jobs, and let S_j be the feasible half-hour-aligned start candidates for job j. The binary decision variable x_{js} equals 1 if job j starts at candidate s, and 0 otherwise. Let e_{js} denote the estimated emissions for job j when it starts at s, calculated by allocating its constant-power energy across every carbon-intensity interval overlapped by its runtime. The objective is:

$$\text{Minimize: } \Sigma_{j \in J} \; \Sigma_{s \in S_j} \; e_{js} \, x_{js}$$

Each job must be assigned exactly one start time. The candidate set S_j contains only starts at or after the job's arrival and no later than arrival plus its configured max_delay_minutes:

$$\Sigma_{s \in S_j} \, x_{js} = 1 \quad \text{for every job } j \in J$$

For each half-hour slot q, capacity is enforced at the slot level: a job counts toward the slot's CPU demand if it overlaps any part of that slot. This is a conservative slot-level approximation; it does not model capacity continuously at every instant. For tested capacity K (8, 12, or 16 cores), the constraint is:

$$\Sigma_{j \in J} \; \Sigma_{s \in S_j} \; a_{jsq} \, c_j \, x_{js} \leq K \quad \text{for every slot } q$$

Here, c_j is the number of CPU cores required by job j, and a_{jsq} equals 1 if the execution of job j starting at s overlaps any part of slot q, and 0 otherwise. The minimum-delay comparator uses the same feasible set and capacity constraints but minimizes total job delay instead of emissions.

Each CPU core is assumed to draw a constant 50 W throughout its runtime. This is an analyst's assumption, not a measured value, and it omits idle power, load-dependent power, cooling, network, storage, and embodied emissions. Only operational emissions are reported; no Software Carbon Intensity score or per-functional-unit figure is produced [4].

3.3 Data Analysis

Energy consumption is estimated from CPU-core requirement, assumed power per core, and runtime. For a job requiring c CPU cores and running for t minutes, estimated energy is:

$$E = [c \times 50 \times (t / 60)] / 1000$$

where E is energy in kWh. For emissions accounting, the job's runtime is split across the 30-minute intervals it overlaps. Under the constant-power assumption, energy assigned to interval i is proportional to the number of job-running minutes overlapping that interval. Carbon emissions are calculated by multiplying interval energy by the corresponding intensity and summing across the runtime:

$$C = \Sigma_i \, (E_i \times I_i)$$

where C is total carbon emission, E_i is the energy consumed during interval i, and I_i is the carbon intensity during that interval.

$$S = C_baseline - C_carbon\text{-}aware$$

The saving is the arrival-time baseline minus the carbon-aware total, S = C_baseline − C_aware, expressed as a percentage by [(C_baseline − C_aware) / C_baseline] × 100. Each experiment changes one parameter while the others stay fixed. The analysis is descriptive and deterministic: no inferential tests are performed, and the scenario comparisons reported here are neither statistical evidence nor guaranteed operational savings.

4. Results

4.1 Effect of Scheduling Flexibility

Figure 1 shows estimated savings rising as the flexibility window widens. Across the seven dates the baseline totals 1,599.81 gCO_2, and the savings for 0, 30, 60, 120, and 180 minutes are 0%, 0.89%, 1.50%, 2.41%, and 4.21% respectively. A wider window simply gives the scheduler more candidate launch times.

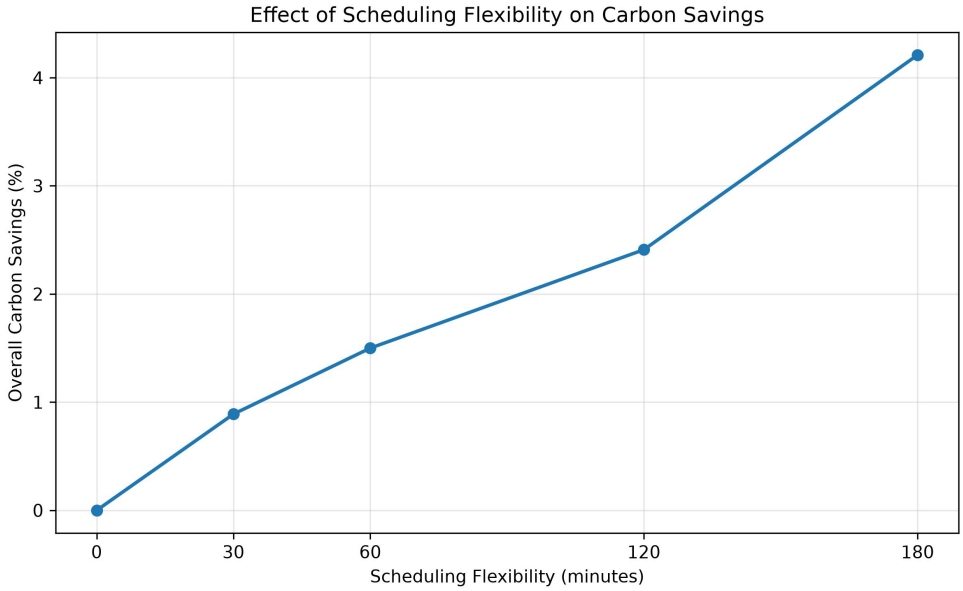


Figure 1. Estimated carbon savings as a function of scheduling flexibility.

4.2 Daily Variation in Carbon Savings

At 180 minutes, daily savings range widely, from roughly 0.01% to 9.68%, with an unweighted mean of 3.57%. The 4.21% value, by contrast, comes from the summed emissions of the seven daily runs, so it is baseline-weighted and differs from the plain daily average. This spread indicates that the benefit follows each day's intensity pattern rather than the flexibility setting alone.

Table 2. Complete daily results at 180 minutes of global flexibility.

Date (IST)	Baseline (gCO_2)	Carbon-aware (gCO_2)	Saved (gCO_2)	Saving (%)
2026-08-24	327.325	313.675	13.650	4.170%
2026-08-25	150.217	143.200	7.017	4.671%
2026-08-26	209.017	208.667	0.350	0.167%
2026-08-27	209.167	207.567	1.600	0.765%
2026-08-28	329.717	297.792	31.925	9.683%
2026-08-29	143.108	143.092	0.017	0.012%
2026-08-30	231.258	218.517	12.742	5.510%

4.3 Sensitivity to Carbon-Intensity Profiles and Workload Size

Figure 2 reports the comparison at 60 minutes of flexibility. The real profile yields an estimated saving of 3.46 gCO_2 (1.50%), dropping from 231.26 to 227.80 gCO_2. Scaling every intensity value by 1.5 raises the totals to 346.89 and 341.70 gCO_2 and the saving to 5.19 gCO_2, while the percentage stays at 1.50% — a proportionality check rather than independent evidence of robustness. Reversing the order of the values gives 191.68 and 183.50 gCO_2, a saving of 8.18 gCO_2 (4.27%). That reversed series is a synthetic counterfactual used only to isolate the effect of ordering; it is not a second observed week and does not show that such a profile is realistic.

For the smaller J1–J5 subset the estimated saving is 2.23 gCO_2 (1.98%). Because this subset performs a different amount of work, its percentage is a fairer basis for comparison than its absolute grams.

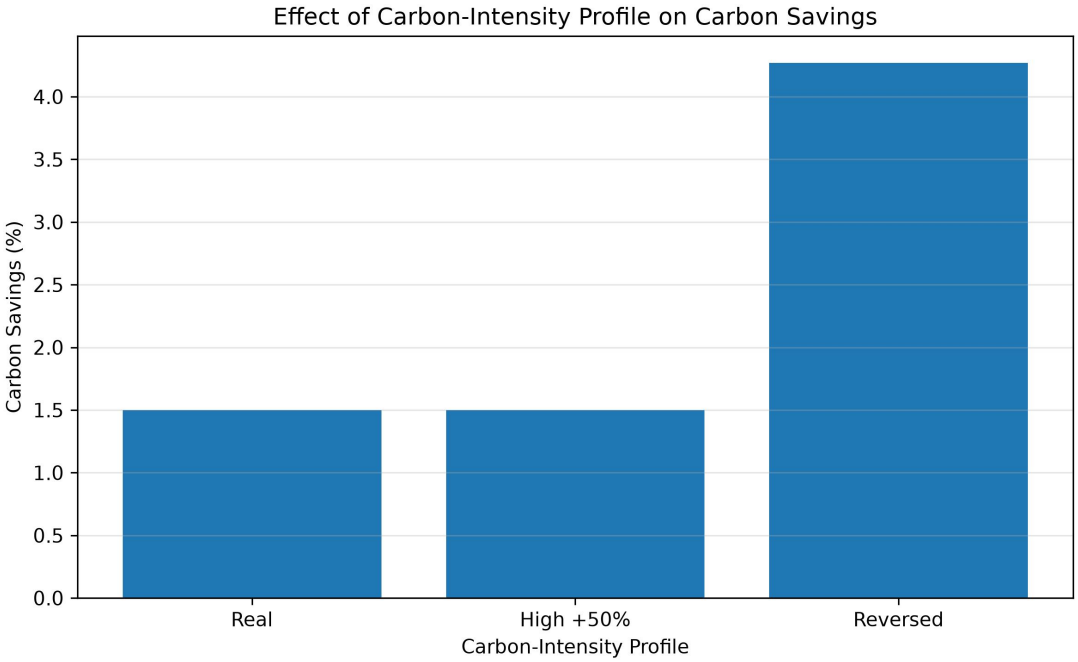


Figure 2. Estimated carbon savings under real, uniformly increased, and reversed carbon-intensity profiles.

Table 3. Summary of estimated carbon savings across selected experimental scenarios.

Experiment	Scenario	Carbon saved (gCO_2)	Saving (%)
Flexibility	30 min	14.27	0.89
Flexibility	60 min	23.98	1.50
Flexibility	120 min	38.53	2.41
Flexibility	180 min	67.30	4.21
Daily variation	Mean daily saving, 180 min	—	3.57
Carbon profile	Real profile, 60 min	3.46	1.50
Carbon profile	Intensity scaled ×1.5, 60 min	5.19	1.50
Carbon profile	Reversed profile, 60 min	8.18	4.27
Workload size	Small workload J1–J5, 60 min	2.23	1.98

Note: Absolute savings are reported in gCO_2. The flexibility rows aggregate the seven-day experiment; profile and workload-size rows come from separate experiments with different emissions baselines. The daily mean is a mean of daily percentages, not an aggregate absolute saving.

4.4 Sensitivity to Workload Configuration

Figure 3 shows how estimated savings respond to workload configuration. When runtimes are varied uniformly, the percentage saved falls from around 1.97% at 15 minutes to about 1.02% at 120 minutes: a short job fits neatly into a clean interval, while a long one spans many intervals and can relocate only part of its load. Changing core counts under the fixed 50 W-per-core rule leaves the percentage almost unchanged at about 1.60%, because energy and saving grow together with cores — only the absolute grams move. Increasing the uniform delay tolerance from 15 to 180 minutes raises the estimate from 0% to roughly 4.21%. All of these results apply only to the model and scenarios tested.

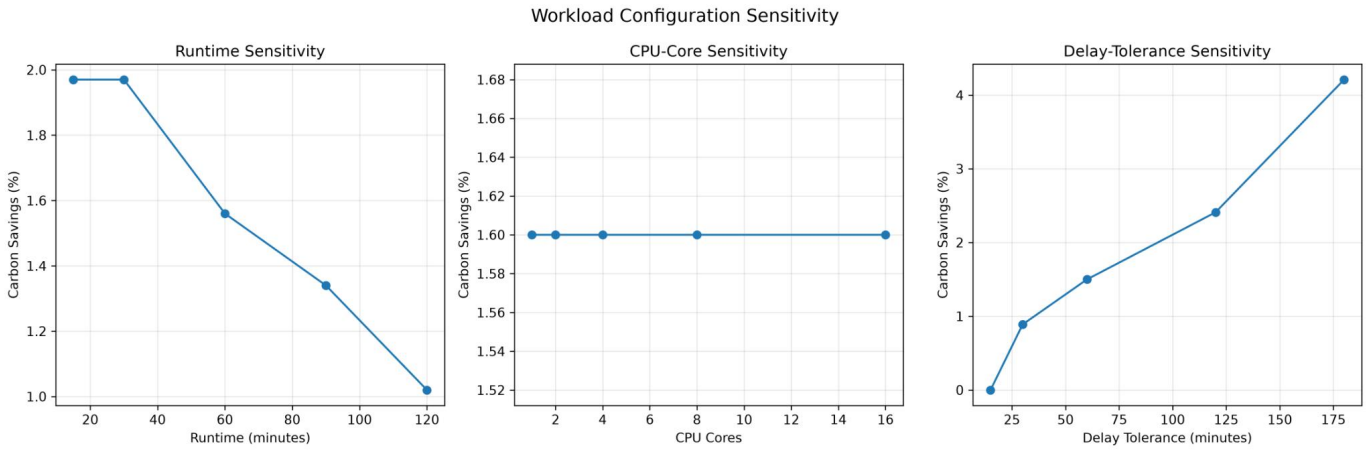


Figure 3. Sensitivity of estimated carbon savings to runtime, CPU-core requirement, and uniform delay tolerance.

4.5 Comparison with a deadline- and capacity-feasible scheduler

To check whether the independent-job result survives real constraints, a mixed-integer linear program (MILP) gives each job exactly one half-hour-aligned start between its arrival and arrival plus that job's max_delay_minutes, ensuring that the combined cores of overlapping jobs stay within the tested capacity in every slot. One objective minimises total estimated emissions under the same 50 W/core model, while a second MILP minimises total delay over the identical feasible set to provide an earliest-feasible comparator. All solves used SciPy's milp interface with HiGHS 1.8.0 (SciPy 1.17.0), a 60-second limit per solve, and a relative gap target of 0.0, and every solve succeeded. Because actual intensity drives the objective, the benchmark remains an oracle rather than a forecasting scheduler.

Table 4. Seven-date aggregate comparison of independent-job and feasible schedules.

Scheduling method	Shared 16-core capacity	Per-job max delay	Emissions (gCO_2)	Savings vs immediate baseline
Immediate-arrival baseline	Not enforced by scheduler; observed schedule is feasible at 16 cores	Arrival time	1,599.81	0.00%
Earliest-feasible schedule	Enforced	Enforced	1,599.81	0.00%
Independent-job carbon oracle	Not enforced	Enforced	1,554.06	2.86%
Carbon-aware feasible MILP	Enforced	Enforced	1,558.69	2.57%

At 16 cores the capacity-feasible plan emits 1,558.69 gCO_2 across the seven daily repetitions, saving 41.12 gCO_2 (2.57%) relative to the immediate-arrival baseline, whereas the deadline-limited independent-job oracle emits 1,554.06 gCO_2 (2.86%). The 4.63 gCO_2 — 0.29 percentage-point — gap is specific to this workload, capacity, and intensity trace. Since the immediate-arrival plan is feasible at 16 cores, it also serves as the earliest-feasible comparator at that capacity.

Figure 4 presents capacity sensitivity at 8 and 12 cores, each compared against the earliest-feasible plan at the same capacity. The carbon-aware MILP reduced estimated emissions by 42.26 gCO_2 (2.52%) at 8 cores, 31.93 gCO_2 (1.99%) at 12 cores, and 41.12 gCO_2 (2.57%) at 16 cores. The 8-core totals are

1,678.81 (earliest-feasible) and 1,636.55 (carbon-aware) gCO_2, while the 12-core totals are 1,601.26 and 1,569.33 gCO_2. Because the 8-core carbon-aware plan remains 2.30% above the unconstrained baseline, which cannot be scheduled at 8 cores, the earliest-feasible plan is the appropriate like-for-like reference at each capacity. The benefit varies with capacity in this scenario, but no general trend is claimed.

Table 5. Seven-date capacity sensitivity relative to the earliest-feasible schedule.

Capacity (cores)	Earliest-feasible (gCO_2)	Carbon-aware MILP (gCO_2)	Saved (gCO_2)	Saving (%)
8	1,678.81	1,636.55	42.26	2.52%
12	1,601.26	1,569.33	31.93	1.99%
16	1,599.81	1,558.69	41.12	2.57%

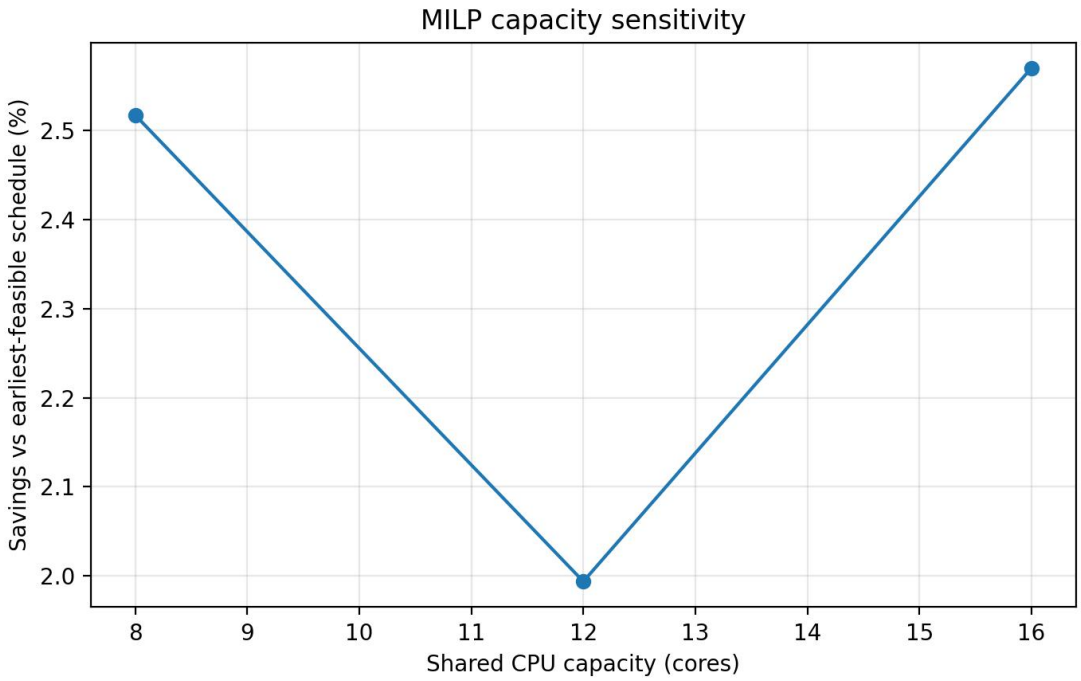


Figure 4. Estimated carbon savings relative to the earliest-feasible schedule at 8-, 12-, and 16-core capacities.

5. Discussion

These results address the paper's central question: estimated carbon savings depend strongly on scheduling flexibility and on the temporal ordering of carbon intensity. They rise from 0% to 4.21% as flexibility grows, but the daily values vary widely, so any one day or flexibility setting captures only part of the picture.

The profile experiment separates the magnitude of intensity from its ordering. Scaling every value by 1.5 scales both absolute emissions and absolute savings while the percentage stays fixed — an algebraic consequence of the proportional objective, not evidence of resilience. Reordering the values changes which ones coincide with candidate start times and therefore changes the percentage; this is a controlled sensitivity probe, not a statement about real grids. Likewise, the near-constant percentage across core counts follows from the assumed linear core-to-power relationship and should not be expected on servers with idle draw and nonlinear utilisation.

The key comparison sets the deadline-limited independent-job oracle against the capacity-feasible MILP. The 4.21% figure is an unconstrained timing benchmark; once per-job delays and a 16-core ceiling are imposed, the estimate falls to 2.57%, compared with 2.86% for the oracle — a capacity cost of about 0.29

percentage points in this case. This is a scenario-specific result rather than a general estimate of capacity-constraint penalties, and every number depends on the synthetic workload, the incomplete seven-date series, the fixed 50 W/core assumption, and hindsight access to actual intensity.

6. Conclusion

A worthwhile next step would be to repeat the study over longer, gap-free records and with the carbon-intensity forecasts a scheduler would really have when making decisions. Power should be measured on physical hardware instead of assumed, and the experiments should extend to larger or trace-derived workloads and to capacities beyond the 8-, 12-, and 16-core settings tested here.

Limitations

The main limitations are the incomplete seven-date record, the retrospective use of actual intensity, the fixed and uncalibrated 50 W-per-core value, the synthetic ten-job workload, and the lack of measured hardware energy. The first scheduler enforces neither individual delays nor shared capacity; the added MILP enforces per-job delay and shared capacities of 8, 12, and 16 cores, yet it still ignores heterogeneous hardware, queueing overhead, migration cost, forecast error, and other resource constraints. The Great Britain signal is national rather than cloud-region-specific, and although the 12 missing timestamps fall outside the intervals used here, the source series remains incomplete. We do not claim that these percentages transfer to deployed services.

Data and code availability

The UK Carbon Intensity API is the stated source of the actual-intensity observations. A reproducibility package accompanies this manuscript and contains the workload and carbon-intensity CSVs, the collection and analysis scripts, an interval-coverage audit, the recomputed daily and aggregate results, and the regenerated figures and tables. The package is available as a reviewer bundle with this manuscript; it has not been deposited in a public repository or permanent archive, and no public URL or DOI is claimed.

References

[1] Radovanović, A., Koningstein, R., Schneider, I., Chen, B., Duarte, A., Roy, B., Xiao, D., Haridasan, M., Hung, P., Care, N., Talukdar, S., Mullen, E., Smith, K., Cottman, M., & Cirne, W. (2023). Carbon-aware computing for datacenters. IEEE Transactions on Power Systems, 38(2), 1270–1280. https://doi.org/10.1109/TPWRS.2022.3173250

[2] Asadov, N., Coroamă, V. C., Franzil, M., Galantino, S., & Finkbeiner, M. (2025). Carbon-aware spatio-temporal workload shifting in edge–cloud environments: A review and novel algorithm. Sustainability, 17(14), 6433. https://doi.org/10.3390/su17146433

[3] Bashir, N., Gohil, V., Belavadi, A., Shahrad, M., Irwin, D., Olivetti, E., & Delimitrou, C. (2024). The sunk carbon fallacy: Rethinking carbon footprint metrics for effective carbon-aware scheduling. Proceedings of the ACM Symposium on Cloud Computing (SoCC '24). https://doi.org/10.1145/3698038.3698542

[4] Green Software Foundation. (2024). Software Carbon Intensity (SCI) specification (Version 1.1). https://sci.greensoftware.foundation/

[5] National Energy System Operator. (n.d.). Carbon Intensity API documentation. https://api.carbonintensity.org.uk/

[6] Kaur, N., Bansal, S., & Bansal, R. K. (2021). Survey on energy efficient scheduling techniques on cloud computing. Multiagent and Grid Systems, 17(4), 351–366. https://doi.org/10.3233/MGS-220357

[7] Verma, P., Maurya, A. K., & Yadav, R. S. (2024). A survey on energy-efficient workflow scheduling algorithms in cloud computing. Software: Practice and Experience, 54(5), 637–682. https://doi.org/10.1002/spe.3292

[8] Biswas, D., Jahan, S., Saha, S., & Samsuddoha, M. (2024). A succinct state-of-the-art survey on green cloud computing: Challenges, strategies, and future directions. Sustainable Computing: Informatics and Systems, 44, 101036. https://doi.org/10.1016/j.suscom.2024.101036

[9] Maji, D., Bashir, N., Irwin, D., Shenoy, P., & Sitaraman, R. K. (2024). The Green Mirage: Impact of location- and market-based carbon intensity estimation on carbon optimization efficacy. In Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems (E-Energy '24). ACM. https://doi.org/10.1145/3632775.3639587

[10] Wiesner, P., Behnke, I., Scheinert, D., Gontarska, K., & Thamsen, L. (2021). Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. In Proceedings of the 22nd International Middleware Conference (Middleware '21) (pp. 260-272). ACM. https://doi.org/10.1145/3464298.3493399

[11] Hanafy, W. A., Liang, Q., Bashir, N., Irwin, D., & Shenoy, P. (2023). CarbonScaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 7(3), Article 57. https://doi.org/10.1145/3626788

[12] Hanafy, W. A., Wu, L., Irwin, D., & Shenoy, P. (2025). CarbonFlex: Enabling carbon-aware provisioning and scheduling for cloud clusters. arXiv:2505.18357. https://doi.org/10.48550/arXiv.2505.18357

[13] Piontek, T., Haghshenas, K., & Aiello, M. (2024). Carbon emission-aware job scheduling for Kubernetes deployments. The Journal of Supercomputing, 80, 549-569. https://doi.org/10.1007/s11227-023-05506-7

[14] Bahreini, T., Tantawi, A. N., & Youssef, A. (2023). A carbon-aware workload dispatcher in cloud computing systems. In 2023 IEEE 16th International Conference on Cloud Computing (CLOUD) (pp. 212-218). IEEE. https://ieeexplore.ieee.org/document/10255008

[15] Bahreini, T., Tantawi, A. N., & Tardieu, O. (2024). Caspian: A carbon-aware workload scheduler in multi-cluster Kubernetes environments. In 32nd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) (pp. 1-8). IEEE. https://ieeexplore.ieee.org/document/10786568

[16] Breukelman, E., Hall, S., Belgioioso, G., & DÃ¶rfler, F. (2024). Carbon-aware computing in a network of data centers: A hierarchical game-theoretic approach. In 2024 European Control Conference (ECC) (pp. 798-803). IEEE. https://ieeexplore.ieee.org/document/10591261

[17] Hall, S., Micheli, F., Belgioioso, G., RadovanoviÄ‡, A., & DÃ¶rfler, F. (2024). Carbon-aware computing for data centers with probabilistic performance guarantees. arXiv:2410.21510. https://doi.org/10.48550/arXiv.2410.21510

[18] Schweisgut, D., Benoit, A., Robert, Y., & Meyerhenke, H. (2025). Carbon-aware workflow scheduling with fixed mapping and deadline constraint. In Proceedings of the 54th International Conference on Parallel Processing (ICPP '25) (pp. 627-637). ACM. https://doi.org/10.1145/3754598.3754673

[19] Yang, J., Saad, Z., Wu, J., Niu, X., Leung, H., & Drew, S. (2025). A survey on task scheduling in carbon-aware container orchestration. arXiv:2508.05949. https://doi.org/10.48550/arXiv.2508.05949

[20] Den Toonder, J., Braakman, P., & Durieux, T. (2024). S.C.A.L.E: A CO2-aware scheduler for OpenShift at ING. In Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24) (pp. 429-439). ACM. https://doi.org/10.1145/3663529.3663862