# KubeFlex: A Carbon-Aware Scheduler with Live Migration

Sallar Farokhi
*Sustainable Systems Lab*
*University of California, Santa Cruz*
Santa Cruz, CA
sfarokhi@ucsc.edu

Abel Souza
*Sustainable Systems Lab*
*University of California, Santa Cruz*
Santa Cruz, CA
asouza@ucsc.edu

***Abstract*—Cloud computing is a business that rents its computing services to businesses and developers alike. To satisfy a client's desire for edge computing, such a business would be tasked with running a variety of workloads simultaneously, across whatever company hardware is provided. As these industry-level workloads occur at scale, scheduling techniques can help the cluster optimize for costs; running processes on the smallest hardware possible can optimize for power.**

**Yet, this one decision made at the OS level can have major environmental implications. While carbon-aware scheduling has emerged as an important research direction, most existing systems focus on initial placement policies without supporting dynamic workload migration during execution.**

**This paper presents the architectural design and implementation of KubeFlex, a carbon-aware Kubernetes scheduler with live-migration capabilities for containerized workloads. The system implements three distinct scheduling policies representing different optimization strategies: static initial placement, reactive hourly migration, and forecast-based migration. We identify technical challenges inherent to CRIU-based migration, characterize system limitations and compatibility constraints, and propose architectural enhancements required for production deployment. This work establishes a foundation for integrating environmental metrics into cluster-level scheduling, while providing practical guidance on the architectural and technical challenges of live container migration.**

***Index Terms*—Scheduling, Kubernetes, Live Migration, Docker, Distributed Systems, Cluster Orchestration**

## I. Introduction

The environmental impact of computing infrastructure has become a critical concern as data centers continue to demand ever-increasing portions of power. The need for electricity is driven by the hardware's demand for power, and this can come from a variety of sources. Carbon intensity—the emissions produced per unit of electricity—varies substantially across geographic regions and throughout the day, reflecting the underlying composition of regional power grids. These geographic variations in carbon intensity create an opportunity for workload placement strategies to minimize environmental impact, without sacrificing computational capability.

Traditional cluster schedulers prioritize resource utilization, performance, and cost, while remaining largely indifferent to environmental metrics. However, emerging research and industry initiatives have demonstrated that incorporating carbon awareness into scheduling decisions can yield measurable environmental benefits. The challenge lies in designing systems that can dynamically optimize workload placement using carbon forecasts, all while maintaining service availability and performance guarantees.

This paper presents KubeFlex, a carbon-aware Kubernetes scheduler that integrates three core capabilities: regional carbon intensity forecasting, policy-based scheduling logic to leverage forecasts, and live migration support. KubeFlex is packaged as a group of Kubernetes resources that work in tandem to enable dynamic workload movement without service interruption. The system moves beyond static initial placement by enabling runtime migration decisions that continuously optimize for carbon intensity throughout the forecast. With KubeFlex, we provide a comprehensive analysis of the technical challenges and constraints of CRIU-based container migration in a cloud environment, and a clear identification of research directions for future improvements.

## II. Background

### *Cluster Scheduling and Orchestration*

State-of-the-art cluster management systems, such as Google's Borg [6], have established foundational patterns for large-scale cluster scheduling and resource management, demonstrating approaches for efficient utilization of infrastructure at massive scale. Kubernetes [5] has emerged as one of the dominant open-source orchestration platforms, providing standardized abstractions for containerized workload management across heterogeneous infrastructure. These systems have proven effective at optimizing traditional metrics, such as resource utilization, application performance, and operational costs. However, traditional schedulers focus primarily on these business-centric objectives and typically lack mechanisms for environmental optimization, leaving untapped potential for reducing the carbon footprint of distributed computing.

### *Carbon-Aware Computing*

Recent work has explored integrating environmental metrics into computing decisions at multiple levels of the stack. Services such as Electricity Maps [1] provide real-time and forecasted carbon intensity data for geographic regions, making environmental data accessible to application developers and infrastructure operators.

Carbon intensity, the measure of greenhouse gases emitted over total electricity generated, serves as the standardized metric for measuring the carbon footprint of energy consumption. It's measured as a product of power consumption and other emission factors, such as the usage of fossil fuels, renewable sources, and import/export rates of electricity. In addition, large-scale seasonal patterns and neighboring grids can affect the final carbon intensity of the region. Prior research has examined carbon awareness at the application level [4], enabling individual applications to adapt their behavior based on regional carbon intensity. However, integration of environmental metrics at the cluster scheduling level—where infrastructure management systems make placement decisions affecting thousands of workloads—remains largely unexplored, representing a significant opportunity for environmental impact reduction at scale.

### *Container Migration and Live Migration*

CRIU (Checkpoint/Restore In Userspace) [2] enables process-level state preservation and migration by capturing the complete details and state of a running process. As a result, this technology has the capability to migrate running processes across separate computers. Despite the existence of CRIU, live migration—moving applications between nodes without service interruption—remains a challenging problem in cloud environments, due to the need to preserve complex application semantics and maintain network connectivity [3]. In addition to the issue of network connections, CRIU is only supported on Linux OSes, and is only operable under root privileges, making it unsuitable for a production environment. While checkpoint-restore technology hasn't been fully explored yet, KubeFlex takes the first step towards a system can transport running processes across arbitrary hardware.

## III. System Architecture

KubeFlex consists of several integrated components that collaborate to observe cluster state, forecast carbon intensity across regions, and make dynamic scheduling decisions. As shown in Figure 1, The KubeFlex system is distributed across several worker nodes in a KIND (Kubernetes In Docker) cluster. The architecture follows a modular design philosophy, enabling independent development and testing of each component, while maintaining clear interfaces for coordination.

The system's workflow begins when workloads are submitted to the Kubernetes cluster. At an hourly interval (smallest margin within carbon forecast), the controller system queries the metadata service for carbon intensity forecasts, and applies the configured scheduling policy to determine optimal pod placements. When migration is warranted, the controller instructs the migration service to coordinate the movement of workloads across nodes and regions. This closed-loop system continuously adapts to changing carbon conditions while maintaining cluster stability.

### *Controller Service*

The controller service represents the central component responsible for managing the scheduling policy, timing decisions, and migration coordination. The controller maintains real-time awareness of the cluster's state by querying the Kubernetes API for information like node availability, pod locations, and resource utilization. It periodically interfaces with the metadata service to obtain carbon intensity forecasts for all managed regions, and applies the configured scheduling policy to determine optimal pod placements. When the controller determines that a workload would benefit from migration, it instructs the migration service to execute the migration process. The controller can operate across both simulated and real time modes, supporting historical evaluation for research purposes, as well as live deployment.

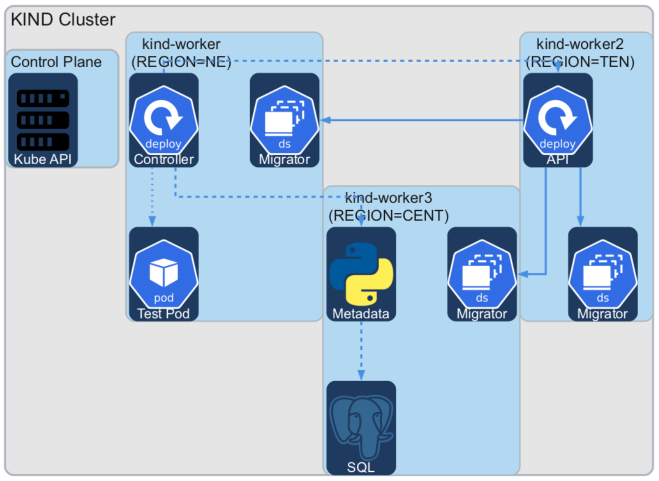

Fig. 1. KubeFlex Architecture

### *Migration Service and Agents*

The migration service provides distributed orchestration and execution of live migration operations. Shown in Figure 1, when the controller determines that a workload should be migrated, it communicates with the migration service, specifying the source pod, destination node, and target region. The migration service coordinates with migration agents deployed on cluster nodes to execute the migration workflow. These specialized migration agents possess elevated privileges enabling them to interface directly with the container runtime, accessing process state, namespaces, and system resources. The agents execute the migration steps in sequence: discovering the container's state, creating checkpoints, transferring state across nodes, and restoring the container at the destination node.

### *Metadata Service*

The metadata service provides carbon intensity forecasts accessible to the controller for scheduling decisions. This service generates forecasts by querying historical and projected carbon intensity data, combining information across all regions of interest. The metadata service presents forecast data as regional time-series projections, enabling the controller to compare expected carbon intensity across regions over configurable time horizons. This interface abstracts away the complexity of forecast generation, exposing only the scheduling-relevant information the controller requires.

### *Database Service*

The database service manages persistent storage of historical carbon intensity data required for forecast generation. Provided by Electricity Maps [1], this Postgres database stores carbon intensity measurements indexed by geographic region and time, enabling efficient retrieval of historical patterns and aggregation for forecast generation. This service provides the foundation for KubeFlex's forecasting capability.

## IV. Scheduling Decision Framework

### *Cluster Design and Regional Distribution*

The system is designed around a Kubernetes cluster with nodes distributed across multiple geographic regions. Each node is labeled with its assigned region, creating a logical mapping between Kubernetes infrastructure and physical power grid regions. Within this model, scheduling decisions translate directly into regional assignments—placing a pod on a node in the Northeast region directs that workload to the Northeast power grid region. The architecture supports variable cluster sizes and node configurations, with minimal requirements including at least three regional zones to enable meaningful migration decisions. Typical deployment configurations feature three to five regions, representing major power grids within North America or other geographic markets. In the KubeFlex example, we use carbon emission data from regions in the Tennessee Valley Authority to the Central US Grid. These are major providers of electricity that span across the different corners of the United States and rely on different means of energy production, such as coal, gas, and nuclear power. Benchmarks in Section IV will demonstrate how carbon intensity varies through the day, and how these different regions can yield varying results across different scheduling methods.

Migration agents must be deployed on every node capable of hosting workloads that can be migrated. These agents require elevated privileges to access the container runtime state and execute checkpoint operations. The privileged access requirement reflects the fundamental nature of process migration—capturing and restoring complete process state necessitates kernel-level operations, which are unavailable to unprivileged containers. This design choice represents a security trade-off that production systems would need to address through careful capability management and isolation.

### *Scheduling Policies*

KubeFlex implements three distinct scheduling policies representing different optimization strategies and complexity levels. These policies provide increasing sophistication in how they balance migration overhead against environmental benefit. The metadata service queries the database for historical records and projects forward in time, creating time-series forecasts of expected carbon intensity for each region over the scheduling horizon. The scheduling decision logic then compares these forecasts across regions to identify which region is

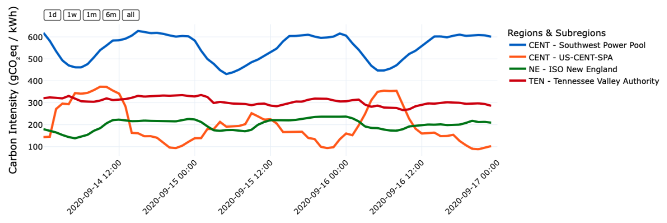

Fig. 2. Average Carbon Intensity for three US regions used in our experiments: Various temporal and spatial variations due to different electricity source demand and supply patterns

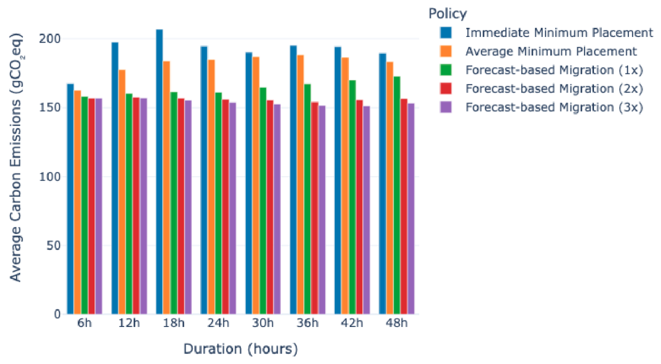

Fig. 3. Average Carbon Emissions by Workload Duration

expected to have minimum cumulative carbon emissions over the workload's expected duration. For the reactive and initial placement policies, forecast generation is simpler, requiring only current or recent historical data, rather than projection into the future.

The first policy, termed Immediate Minimum Placement, assigns workloads to the region with minimum carbon intensity at submission time. Once placed, pods remain on their assigned node throughout their lifetime with no subsequent migration. This policy serves as a baseline for evaluating the benefits and overhead of dynamic migration. It represents the simplest possible carbon-aware approach, providing a control point for measuring migration value.

The second policy, termed Average Minimum Placement, assesses which region will have the lowest carbon intensity on average, across the expected duration of the workload, and places the pod there with no further migrations. The Average Minimum Policy is appropriate for workloads where the carbon savings are not enough to justify migrating the process, such as short-term user access or minimal workloads. Figure 2 displays carbon intensity from the regions used in the KubeFlex cluster. We select forecast patterns from suppliers like the TVA and the Southwest Power Pool because their power production comes from a variety of natural and renewable resources, and they're top contenders for the regions with the lowest emissions. The reputation and performance of these regions suit the benchmarking goals of this project.

Observing the data in Figure 2, the Average Minimum Placement would take a workload that's scheduled at 12:00

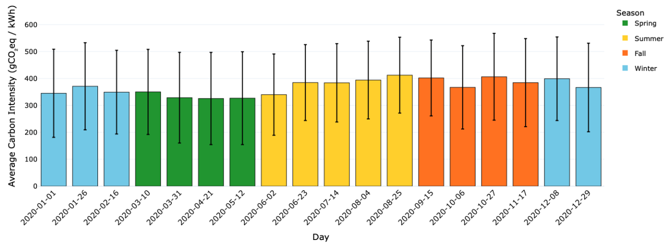

Fig. 4. Average Carbon Intensity by Season

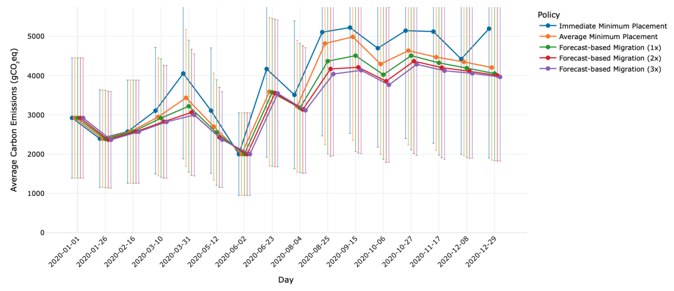

Fig. 5. Yearly Average Carbon Emissions for each Policy

on the 15th, and place it on the Southwest region rather than the New England one. In this regard, the Immediate Minimum Policy sees a sharp decline in efficiency, as shown in Table I.

The third policy, termed Forecast-Based Migration, projects carbon intensity across all regions over the remaining expected duration of each workload, and selects the optimal schedule of regions with lowest cumulative carbon emissions over that time span. This approach makes migration decisions based on expected future conditions rather than the current state, balancing the overhead of migration against anticipated carbon savings. This policy requires workloads to provide the estimated duration, enabling the scheduler to make informed trade-offs between migration cost and environmental benefits.

TABLE I
POLICY EFFICIENCY BY DURATION
BENCHMARK TIME: 09/15/2020, 12:00

| Hours | Imm Lowest | Avg Lowest | 1 Mig | 2 Mig | 3 Mig |
|---|---|---|---|---|---|
| 6 | 87.65 | 97.21 | 98.76 | 99.99 | 100 |
| 12 | 66.71 | 95.79 | 99.04 | 99.99 | 100 |
| 18 | 72.28 | 83.54 | 96.09 | 98.16 | 99.38 |
| 24 | 77.72 | 77.72 | 97.05 | 98.62 | 99.54 |
| 30 | 80.28 | 80.28 | 9602 | 97.47 | 98.73 |
| 36 | 74.08 | 78.54 | 85.47 | 97.76 | 98.88 |
| 42 | 75.11 | 80.37 | 84.77 | 96.82 | 98.11 |
| 48 | 77.73 | 80.52 | 86.53 | 94.37 | 98.30 |

The average carbon emissions are determined by taking the carbon output for each region used in a simulated run, and the Forecast-Based Migration Policy was assessed in three different cases; limiting the scheduler to one, two, and three migrations. While there's a clear correlation between the number of allotted migrations and the sustained efficiency of the policy, the returns diminish as the workload extends past the 24-hour cycle. This is likely due to the data [1], which demonstrates that the minimum carbon region changes between 2-3 times a day across the United States, at the regional and sub-regional scale. Table I reflects the same data from Figure 3, but displays it as a comparison to the optimal scheduling path of the workload, which would be to have the workload running in the region with minimum emissions at all times. Since the Immediate Lowest Policy does not have the opportunity to migrate, the efficiency of the policy dropped below 100% within the first 6 hours, indicating that the minimum region had changed since then.

We see in Figure 3 that the average carbon intensity across the later policies is consistently optimal, well into the first 12 hours of runtime. We assert that as time goes on, the number of required migrations will need to occur proportionally, otherwise all scheduling policies will eventually plateau in expected carbon emissions. This is further corroborated by the later policies in Table 3, where the Forecast-Based Migration Policies begin to deteriorate in efficiency after the 24 hour mark, and the non-migration policies begin to plateau in efficiency loss. Enterprise clusters that wish to implement a carbon-aware scheduler at scale would need to assess the tradeoffs between migration and short-term emissions for shorter jobs. Similarly so, if a process is lightweight and scheduled to run for a long time, one must assess the overhead gained from continuously migrating the pods across different nodes for a minimal decrease in carbon emissions.

We can further observe the data in Figure 3 from a larger scale. Figure 4 displays the average carbon intensity of these policies throughout the year, indicating a slight dip in carbon emission in the springtime, and an uptick in-between fall and summer. Building upon this existing trend, Figure 5 indicates that between the months of August and November, non-migration based policies consistently yield weaker results, indicating a higher number of region changes. Long running workloads can be assessed at the monthly forecast scale, and the Average Lowest Region Policy could prove to be less computationally expensive, for a marginal decrease in carbon efficiency.

By including this carbon forecast data in KubeFlex's scheduling framework, we can implement the Forecast-Based Migration policy across real workloads in a cloud-based setting.

## V. MIGRATION WORKFLOW

In KubeFlex's test implementation, we deploy a KIND cluster with four nodes, one as the control plane and three worker nodes. As mentioned earlier, we label each of the worker nodes with a region from the sample dataset (New England, Tennessee, Southwest), and every hour, the scheduler observes incoming jobs and their metadata, and decides whether or not to invoke the migration service. KubeFlex's migration service occurs across two Linux containers, where the main process thread is captured and transferred in a one-time process to

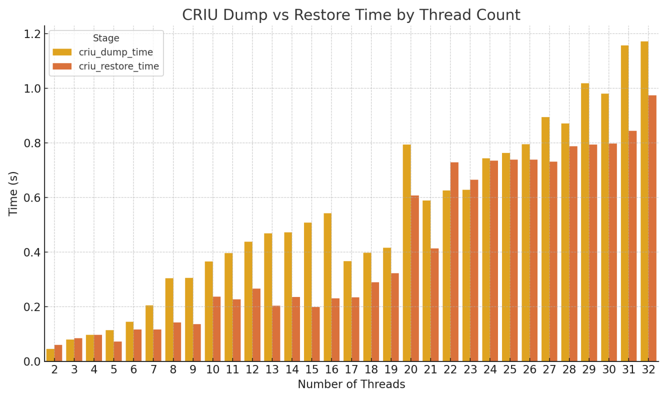

Fig. 6. Checkpoint/Restore Times over Threads

a prepared container in the new node. The process captured is a simple shell script that periodically writes the date to a file. The migration of a workload from a source node to a destination node proceeds through a well-defined sequence of phases, abstracting the complexity of process migration into manageable steps.

TABLE II
MIGRATION TIMELINE

| Time (s) | Event |
|---|---|
| 0.000 | Getting Node Information |
| 0.007 | Node Validation |
| 1.825 | Testing Kind Node Access |
| 2.681 | Getting Source Container Info |
| 2.681 | Getting Container info via K8S API kind-worker3 |
| 2.980 | Creating Target Pod |
| 6.646 | Performing CRIU Dump |
| 7.822 | Transferring Checkpoint |
| 8.360 | Copying Script Data |
| 9.227 | Executing CRIU Restore |
| 10.053 | Final Verification |
| 10.053 | Getting Container Info Via K8S API kind-worker2 |

The validation phase verifies that both source and destination nodes exist and are in operational state, and confirms that the workload is running on the source node, and that the destination node has capacity for the workload. Once validation succeeds, the state capture phase executes on the source node, where the source migration agent captures the complete process state, including memory pages, file descriptors, file mounts, and namespace configurations. This state capture is performed without interrupting normal workload execution, though it does impose computational overhead on the source system.

Figure 6 demonstrates the performance of CRIU's dump/restore command, in comparison to the number of threads being processed. The figure demonstrates a fairly linear overhead for CRIU's runtime, but the restore function sees less of a decline than the dump function. Note that the simplicity of the threads in parallel being migrated may contribute to the results of Figure 6, and that CRIU's performance of checkpointing real, distributed workloads may vary.

During the preparation phase, the destination node is prepared to receive the workload, including the creation of any necessary environment configurations. The migrator services SSH directly into the workload and uses the CRIU library to capture the state of the container's main process. The information is then transferred across the network with the Kubernetes API. The state restoration phase executes on the destination node, where the captured state is restored with CRIU, recreating the workload in an identical state to that at the source node. We migrated a simple thread writing to a log file, and observed that as the process was restored in the new container, the old and new pod would run in parallel with the same contents, except for one difference: the timestamp that would print on every log would be two seconds later in the new pod, indicating that downtime can be negligible.

If migration fails at any phase, the source workload continues running, preserving application continuity and preventing loss of service. This graceful failure mode is essential for production reliability, though it may result in the workload remaining in a suboptimal region if repeated migration attempts fail.

## VI. System Limitations

### *Container Compatibility Challenges*

Live migration through checkpoint-restore requires containers to maintain compatibility with the underlying migration mechanism. Not all containerized applications can be successfully migrated using this approach. Containers must be compatible with CRIU at the Linux kernel level and must not depend on resources unavailable during the restore phase. Certain application categories, including those with complex graphical user interfaces, multimedia streaming operations, or real-time hardware interactions, exhibit poor compatibility with current migration technology.

The success of KubeFlex's migration service is due to the defined structure of the cluster's services. Tailored Dockerfiles allow the pod to properly be encapsulated with minimal problems, but this limitation prevents the current system from serving as a general-purpose container migration solution, and restricts its applicability to broader workload types.

A fundamental limitation of current checkpoint-restore technology is its support for network connections and inter-process communications. When a workload is migrated from one node to another, established TCP connections are lost, and applications that depend on persistent network connections must manually re-implement reconnection logic. This limitation prevents transparent migration of certain classes of applications without additional application-level support. Services maintaining long-lived connections to clients or backend systems cannot migrate without incurring connection failures and requiring client-side retry logic.

### *Platform and Kernel Dependencies*

The checkpoint-restore mechanism is fundamentally dependent on Linux kernel capabilities, which may be unavailable in other operating systems. CRIU requires Linux hosts with specific kernel features and versions. The system requires careful version management to ensure compatibility between the checkpoint-restore tools and the underlying kernel. Compatibility across different Linux distributions and kernel versions is limited, and kernel upgrades or changes in system configuration may affect migration capability. Part of KubeFlex's strength is this ability to copy the pod's Dockerfile as part of the migration process, but even this is insufficient as container builds expand in complexity and diversity.

Complex file system hierarchies with numerous mount points present challenges for migration. The system must discover and transfer all mounted file systems and bind-mount configurations required by a workload. While the migration mechanism can handle standard volume configurations, certain complex mount scenarios may not be correctly preserved across migration. KubeFlex statically declares the file mounts for the container being migrated, and labels them as external in order to reduce system dependencies. This limitation is particularly relevant for workloads with sophisticated storage requirements or dynamic file systems modifications. Workloads that create or modify mount points during execution present particular challenges, as these changes may not be consistently captured in the checkpoint.

## VII. Future Work

### *Enhanced Migration Capabilities*

Future research should address the fundamental technical limitations of checkpoint-restore by investigating alternative migration mechanisms preserving network connections and other application state. Network-level solutions including service mesh integration or overlay networks may enable transparent migration without application modification. Application-aware migration frameworks could preserve application-specific state through coordinated checkpointing between the migration mechanism and application logic. Specialized mechanisms for stateful services such as databases, caches, and persistent services would extend migration capability to critical infrastructure components currently incompatible with checkpoint-restore migration.

### *Production-Scale Systems*

Production deployment across multiple data centers and regions requires support for major cloud providers and heterogeneous infrastructure beyond the homogeneous KIND clusters used for research. Federation of multiple scheduling domains would enable coordination across independently managed clusters. Integration with existing enterprise infrastructure and policies ensures compatibility with organizational requirements. Comprehensive observability and debugging capabilities provide visibility required for operational management. Compliance with security and regulatory requirements ensures suitability for sensitive workloads and regulated industries.

### *Further Scheduling Improvements*

Beyond the three implemented policies, production systems require multi-objective optimization balancing environmental, performance, and cost objectives according to organizational priorities. Machine learning models could predict optimal migration timing and destination selection based on historical patterns and forecasted data. Cost-benefit analysis could compare migration overhead against anticipated savings to determine whether migration should proceed. Dynamic policy frameworks would enable organization-specific optimization strategies, reflecting different environmental commitments and performance requirements.

## VIII. Conclusion

KubeFlex presents a complete architectural framework for carbon-aware scheduling in Kubernetes environments with live-migration capabilities. The modular design cleanly separates scheduling logic, migration orchestration, forecast generation, and data management, enabling independent evolution of each component while maintaining well-defined interfaces. The system demonstrates three scheduling policies representing a spectrum from simple static placement to sophisticated forecasting, enabling comparative evaluation of different optimization strategies and their trade-offs regarding migration overhead versus environmental benefit.

We evaluate the challenges that prevent the migration of homogeneous workloads. These limitations do not invalidate the broader approach to carbon-aware scheduling, but rather establish technical constraints that production systems must address in future implementations.

The ultimate vision of carbon-aware computing requires integrating environmental metrics into operational decisions, while maintaining performance and availability guarantees. KubeFlex establishes a practical platform for exploring this vision and provides clear architectural and technical guidance for advancing the field. As environmental concerns increasingly drive technology decisions, research and development of carbon-aware distributed systems will become central to sustainable computing infrastructure.

## Acknowledgment

The authors thank the Sustainable Systems Lab at UC Santa Cruz for support and guidance throughout this work. We acknowledge the Kubernetes community and CRIU project for foundational technologies. We also recognize the contributions of team members who provided feedback on architectural design and system implementation.

## References

[1] Electricity Maps. Real-time co2 emissions of electricity consumption. Electricity Maps Website, 2021. Accessed: 2025.

[2] Pavel Emelyanov, Alexey Gurtovoy, Andrey Khorenko, and Kirill Kolyshkin. Criu: Checkpoint/restore in userspace. In *Proceedings of the Linux Symposium*, pages 81–89, 2013.

[3] Michael Mirkin, Alexey Kuznetsov, and Kirill Kolyshkin. Containers checkpointing and live migration. In *Proceedings of the Linux Symposium*, pages 85–90, 2008.

[4] Anne-Cécile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys*, 46(4):1–31, 2014.

[5] The Kubernetes Authors. Kubernetes: Production-grade container orchestration. https://kubernetes.io/, 2014. Version latest.

[6] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–17, 2015.