UNIVERSITY OF CALIFORNIA SAN DIEGO

# GreenGrader: A Carbon-Aware Distributed Autograder System

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Computer Science

by

Malcolm Robert McSwain

Committee in charge:

Professor George Porter, Chair
Professor Amy Ousterhout
Professor Aaron Schulman

2023

The Thesis of Malcolm Robert McSwain is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

# DEDICATION

This thesis is dedicated to my dearest great aunt Mary Dunlap, who sadly passed away while I was writing it. A true devotee of academia and computer enthusiast, it is because of her that I chose to pursue research in computer science. She was and always will be an inspiration to me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my profound appreciation and thanks to my research advisor, George Porter. This project would not have been possible without his thoughtful guidance. I would also like to extend this gratitude to the rest of my thesis committee, Pat Pannuto and Amy Ousterhout, who also advised me throughout the duration of this project.

Furthermore, I want to acknowledge the hard work of my fellow researchers Yibo Guo, Joshua Santillan, and Gagan Gopalaiah, who helped immensely with this endeavor. Their contributions truly brought GreenGrader to life.

Finally, I give thanks to my family, friends, and colleagues who supported me along this journey.

ABSTRACT OF THE THESIS

GreenGrader: A Carbon-Aware Distributed Autograder System

by

Malcolm Robert McSwain

Master of Science in Computer Science

University of California San Diego, 2023

Professor George Porter, Chair

GreenGrader is a carbon-aware distributed autograder system designed to minimize the environmental impact of computational workloads. Autograding, the automated assessment of student assignments, is increasingly utilized in computer science education. While valuable pedagogically, it can be resource intensive. GreenGrader aims to minimize the carbon footprint of these workloads through energy-efficient computing and carbon-aware scheduling. It consists of an ingestion pipeline to receive submissions and an execution

pipeline to evaluate them using containers across distributed infrastructure. By integrating with a carbon-aware scheduler and the National Research Platform's HyperCluster, GreenGrader enables geographic workload shifting to optimize carbon emissions. The efficacy of GreenGrader was evaluated using 134 genuine student submissions. Compared to static geographic placement, GreenGrader reduced carbon emissions by 40.91% by shifting workloads based on real-time carbon intensity data, demonstrating the promise of carbon-aware scheduling. Overall, GreenGrader represents an advancement in aligning distributed computing with ecological stewardship. As society advances towards low-carbon systems, GreenGrader provides a model for embedding environmental responsibility within computational workloads.

# INTRODUCTION

The relentless growth of data centers and their insatiable appetite for energy pose significant challenges to our global commitment to sustainability. Accounting for an estimated 70 billion kWh, or nearly 2%, of U.S. electricity consumption, datacenters are emblematic of the larger issue of energy consumption in technology sectors [1]. Within this broad landscape, the field of distributed computing emerges as a microcosm reflecting the broader challenges of carbon emission reduction. Amidst these challenges, optimizing the energy usage of datacenters is an important step toward improving their sustainability.

Autograding plays a vital role in modern education, allowing for the scalable and efficient evaluation of student work. While valuable, the process of conducting this computational evaluation can be surprisingly resource-intensive, prompting a need for solutions that can minimize the environmental impact of these workloads. The challenges of harnessing renewable energy have practical implications for distributed autograding systems. These systems can be designed to operate more sustainably by contributing to overall grid stability through demand-response strategies.

In the quest for carbon efficiency, the virtualization of energy systems has shown promise. The concept of an "ecovisor," which virtualizes the energy system and exposes software-defined control, offers a glimpse into how applications might handle clean energy's unreliability [2]. This tailored control allows each application to manage the unpredictability of clean energy based on its unique requirements, such as control of server power usage and battery charging, paving the way for more carbon-efficient operations. By giving applications control over energy consumption and aligning it with the availability of renewable energy sources, their carbon footprint can be drastically improved.

Simultaneously, the idea of temporal workload shifting has begun to receive attention as a method to reduce emissions. The principle of shifting computational workloads to times when energy supply is less carbon-intensive offers a pathway to further align energy consumption with environmental responsibility [3]. By identifying and analyzing the potential for temporal workload shifting, and through the examination of time constraints, scheduling strategies, and the accuracy of carbon intensity forecasts, this concept has shown promising results in specific regional contexts.

As the dialogue on reducing carbon emissions continues, another promising concept comes to the forefront: geographical workload shifting. This idea involves moving computational tasks to regions where the energy supply is derived from more sustainable and renewable sources, contributing to a significant reduction in carbon emissions. The development of advanced cloud computing and data management technologies now allows for the shifting of computational workloads across different geographical locations, a strategy that can effectively reduce their environmental impact. By utilizing real-time data on energy sources and carbon emissions, computational tasks can be dynamically allocated to regions where their carbon footprint would be minimized. This concept stands as a testament to the potential of workload management in significantly reducing carbon emissions on a global scale. Despite the logistical challenges involved in implementing geographical workload shifting, the potential impact on enhancing carbon efficiency and contributing to global sustainability efforts makes it a compelling avenue for further research and exploration [4].

Drawing inspiration from these insights, this research targets the development of a carbon-aware distributed autograding system. The questions that guide this inquiry are multifaceted: How can a distributed autograding system be constructed to be more effective than

existing solutions like Gradescope? How can such a system meaningfully contribute to reductions in carbon emissions? The objectives are clear and actionable: to build and evaluate a carbon-aware distributed autograding system that institutions can both use and voluntarily contribute computing resources to, and to utilize this system to perform realistic workload tests to determine the efficacy and energy savings of carbon-aware scheduling in this context.

In synthesizing these themes, this research aims to enhance distributed autograding technology with a strong emphasis on carbon awareness. It explores the integration of computational efficiency and environmental impact, reflecting a wider trend towards aligning technology with ecological responsibility. Through the lens of this disciplinary intersection, this thesis contributes to a broader discussion about how we can make our computing systems more sustainable, underlining our shared commitment to the planet's future.

# Chapter 1

# BACKGROUND

## 1.1 Motivation

The digital revolution, coupled with the inexorable advance of technological infrastructure, has given rise to an era where data forms the bedrock of many organizational processes, including those in the academic space. Academic institutions, with their ever-increasing reliance on computational tools for research, simulations, and large-scale educational platforms, have contributed to a surge in datacenter workloads [5]. Datacenters, sprawling complexes laden with servers and network equipment, form the backbone of this digital ecosystem. These centers are strategically located around the globe, with countries like the United States, China, and parts of Europe hosting most of them due to their robust technological infrastructure and market demand [6]. While many are operated by tech giants such as Google, Amazon, and Microsoft, smaller entities, including academic institutions and regional enterprises, also maintain their own datacenters [7]. Their energy needs, primarily for computational processes and crucial cooling mechanisms, emphasize the significance of understanding and optimizing datacenter workloads.

Datacenter workloads refer to the diverse range of tasks or jobs that datacenters are designed to execute, encompassing everything from simple data storage and retrieval operations to more complex computational tasks such as simulations or data analytics. These workloads can vary significantly in their computational requirements, duration, and urgency, and they present a constant challenge in terms of optimal resource allocation. Central to this challenge is the concept of workload scheduling. At its essence, workload scheduling is the strategic assignment

of tasks to specific computational resources within a datacenter, ensuring that operations are executed efficiently and without delay. The objective of this scheduling is not merely to ensure operational efficiency but also to minimize costs, ensure fault tolerance, and maintain consistent service quality [8].

An emerging concept in the realm of workload scheduling is geographical shifting—essentially the relocation of workloads to different datacenter locations based on a myriad of factors, including but not limited to energy costs, resource availability, and network congestion [9]. One pivotal factor driving the importance of geographical shifting is carbon intensity. Carbon intensity is a metric that quantifies the amount of carbon dioxide (CO2) emissions produced per unit of electricity consumed, often measured in grams of CO2 per kilowatt-hour (gCO2/kWh) [10]. The computation of carbon intensity involves analyzing both the sources of energy (renewable versus non-renewable) and the efficiency of the energy production methods in place. As global emphasis on sustainability intensifies, the ability to execute workloads in regions with lower carbon intensity offers a compelling avenue not only for cost savings but also for significant reductions in the environmental footprint of datacenter operations. This alignment of technological advancement with ecological responsibility lays the foundational motivation for this project.

### 1.2 Related Work

The GreenGrader project combines energy-efficient computing, sustainable cloud services, and geographic workload shifting to advance discussions in these important fields. A myriad of research and real-world applications have delved into similar areas, striving to

optimize energy efficiency, minimize carbon emissions, and promote sustainability in computational environments. This section examines various notable works that resonate with the GreenGrader project's objectives and methodologies.

Agarwal et al. (2021) offers a novel perspective on harnessing renewable energy in data centers, aligning with the zero-carbon commitments set forth by prominent cloud service providers. The key challenge identified is the significant variability in the power generated from renewable sources. Traditional mitigation strategies like utilizing batteries or grid transmission are recognized as inadequate due to their scalability issues, overhead, or lack of "green-ness." To address this, the authors introduce the concept of a Virtual Battery (VB). Unlike conventional approaches that attempt to align power availability with computational demand, the VB paradigm shifts computational demand to coincide with the availability of power [11]. CarbonCast is an open-source instrument developed for multi-day carbon intensity forecasts, offering the code and data to the scholarly community freely. This tool facilitates better anticipation of electric grid carbon intensity over several days, which is crucial for data centers aiming to optimize their energy consumption based on carbon emissions [12]. Carbon Explorer is a framework proposed by researchers at Meta, addressing the high-carbon energy consumption in datacenters by optimizing a mix of solutions for 24/7 carbon-free datacenter operation. The optimization is contingent on the geographic location and workload, thus allowing for a more targeted approach in reducing carbon emissions and transitioning towards renewable energy utilization. The framework also delves into balancing the trade-offs between operational and embodied carbon, which is crucial for achieving a more sustainable datacenter operation [13].

One of the most related research domains to GreenGrader is the exploration of geographical workload shifting. This concept involves dynamically allocating computational

tasks to regions where the energy supply is predominantly derived from renewable sources, thereby minimizing the carbon footprint of these operations. Wiesner et al. (2023) scrutinizes the carbon intensity fluctuations in the public power grid, contingent on energy sources and demand. It identifies the characteristics of delay-tolerant workloads and evaluates the potential of temporal workload shifting in various regions. The authors elaborate on shifting computational workloads towards times with lower carbon-intensive energy supply, examining its potential impact across Germany, Great Britain, France, and California during 2020 [14].

Furthering the discourse on geographical workload distribution, Lin et al. (2023) delves into the cost and carbon reduction potential of geographically dispersed data centers with renewable energy integrations. The paper underscores the challenges posed by the stochastic nature of incoming jobs and renewable energy generations, and ventures to explore solutions through carbon-aware load balancing [15]. In the realm of carbon-aware scheduling specifically, the work by Chen et al. (2012) is of significant relevance. The authors propose a holistic workload scheduling algorithm aimed at minimizing the consumption of traditional energy sources, referred to as “brown energy”, across geographically distributed data centers leveraging renewable energy resources [16]. The GreenGrader project aims to integrate a similar scheduling model, in collaboration with a real-world carbon-aware scheduler, implemented as part of a joint research effort by a fellow student in our research group, Yibo Guo [17].

Another notable real-world application of carbon-aware scheduling is embodied in Google's Carbon-Intelligent Computing System (CICS). The CICS is designed to diminish grid carbon emissions from Google’s datacenter electricity usage and to cut operational costs by enhancing resource and power efficiency. The system capitalizes on the temporal flexibility of a significant fraction of Google’s internal workloads that can tolerate delays, provided they are

completed within a 24-hour window. The CICS employs cluster-level Virtual Capacity Curves (VCCs) to set hourly resource usage limits for each data center, optimizing them based on various factors including demand predictions and hourly carbon intensity forecasts. These VCCs, updated daily, set the framework for flexible computing usage, thus reducing compute and power usage during times of anticipated high carbon intensity by delaying certain computing tasks to later, more carbon-efficient hours [18].

The novelty of GreenGrader lies in the application of these concepts to the context of autograding in educational environments. While various projects and research efforts have explored aspects related to the GreenGrader project, the unique combination of energy efficiency, geographical workload shifting, and carbon-aware scheduling within the context of autograding in educational environments is the primary contribution of this thesis.

# Chapter 2

# DESIGN

## 2.1 Initial Considerations

In the formative stages of the GreenGrader project, a major architectural design decision was made to modularize the system for better scalability and maintainability. This entailed strategically decoupling the operations of submission ingestion from its subsequent execution. Consequently, this led to the establishment of two fundamental, yet distinct components within our architecture: the "ingestion pipeline" and the "execution pipeline". The rationale behind this dichotomy was to delineate responsibilities—while the ingestion pipeline was tailored to oversee the seamless transfer of submissions from the Gradescope platform to a centralized database repository, the execution pipeline was vested with the task of fetching these submissions, orchestrating their evaluation within a computational cluster, and subsequently cataloging the results back into the database. The design of both pipelines would prioritize scalability, availability, and reliability as primary considerations.

Diving deeper into the intricacies of the ingestion pipeline (Figure 1), our first foray into the project revolved around an in-depth comprehension of Gradescope's autograding framework. This foundational understanding was critical in identifying the opportune junctures within Gradescope's pipeline where submissions could be gracefully intercepted without compromising the integrity of the process. Upon pinpointing these junctures, our focus then transitioned to engineering a robust mechanism that would facilitate the submission's transit from Gradescope to our ingestion pipeline. At this point, we recognized the need to implement a message queueing system to transport student submissions. After researching various message queue

implementations, we chose the MQTT protocol. This protocol, often used in IoT applications, stood out for its lightweight nature and straightforward machine-to-machine communication. Also central to this mechanism was the need for a meticulously designed database schema—this schema not only stored the submissions themselves but also ensured that associated metadata, crucial for downstream processes, was coherently and securely retained. Among various database implementations, Postgres stood out because of its robustness, scalability, and its track record for reliably handling complex schema designs, making it perfectly suited for our requirements.

Shifting our gaze to the execution pipeline, (Figure 2) a systematic approach was adopted to ensure efficient and accurate processing. The process would be initiated by extracting a submission record from the database, which would be subsequently enqueued into a job management system to await processing. At this point we would harness a black box algorithm, exogenous to this project, with the primary responsibility of allocating a submission to an optimally poised server within our cluster for execution. Our scheduling policy and consequently our definition of "optimal" would vary somewhat depending on the kind of experimentation we were performing, but most of the time our target would represent the server with the lowest carbon intensity. Post-execution, we planned to implement mechanisms to meticulously capture granular details pertaining to the execution—most importantly, the power consumption metrics. Moreover, other vital metadata would be extracted and processed to provide a comprehensive overview of each execution cycle. Concluding this phase, the aggregated results, coupled with the harvested metadata, would be systematically cataloged into a dedicated segment of our database, thus ensuring a structured and cohesive data storage paradigm.

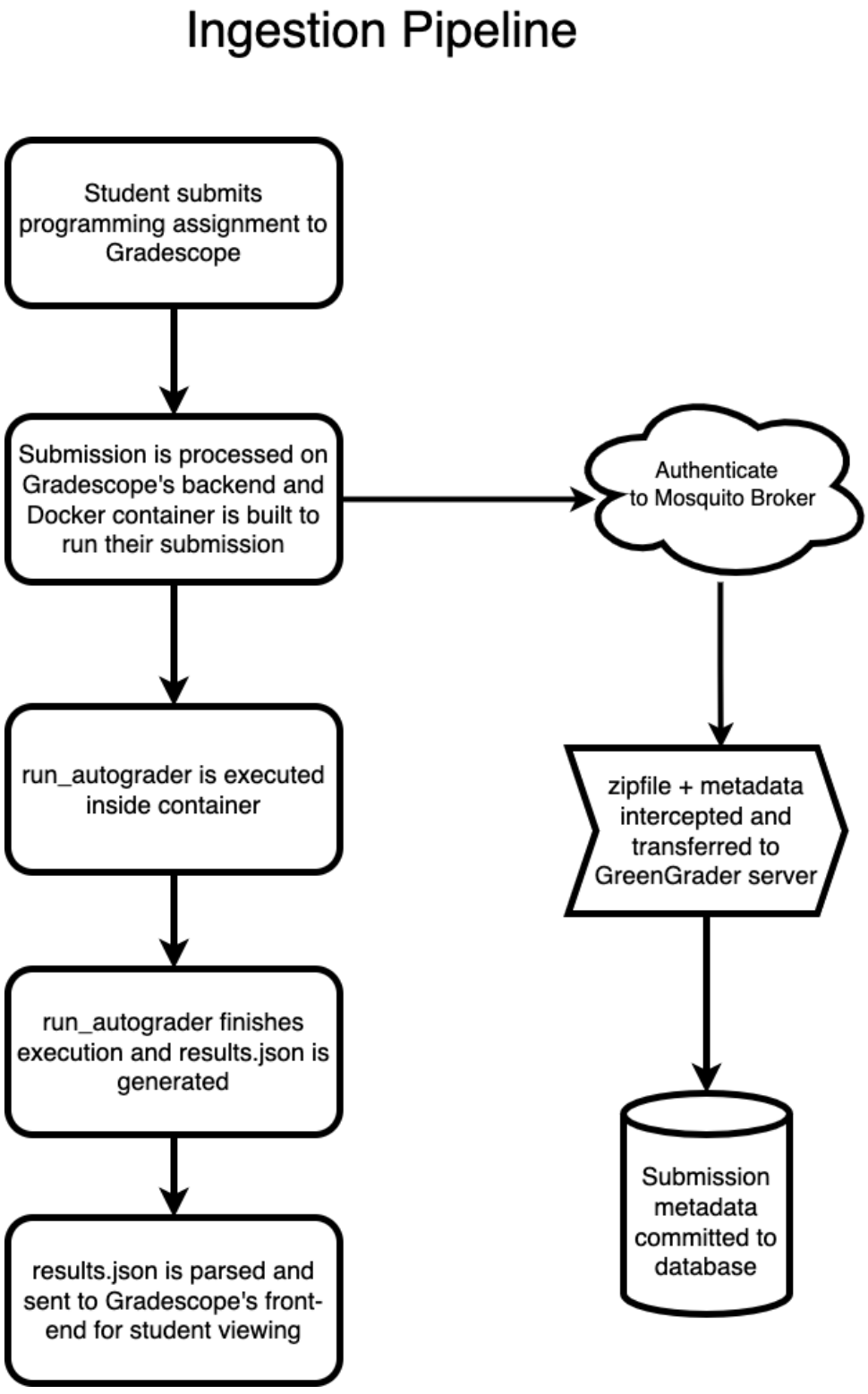


Figure 1: Ingestion Pipeline Diagram

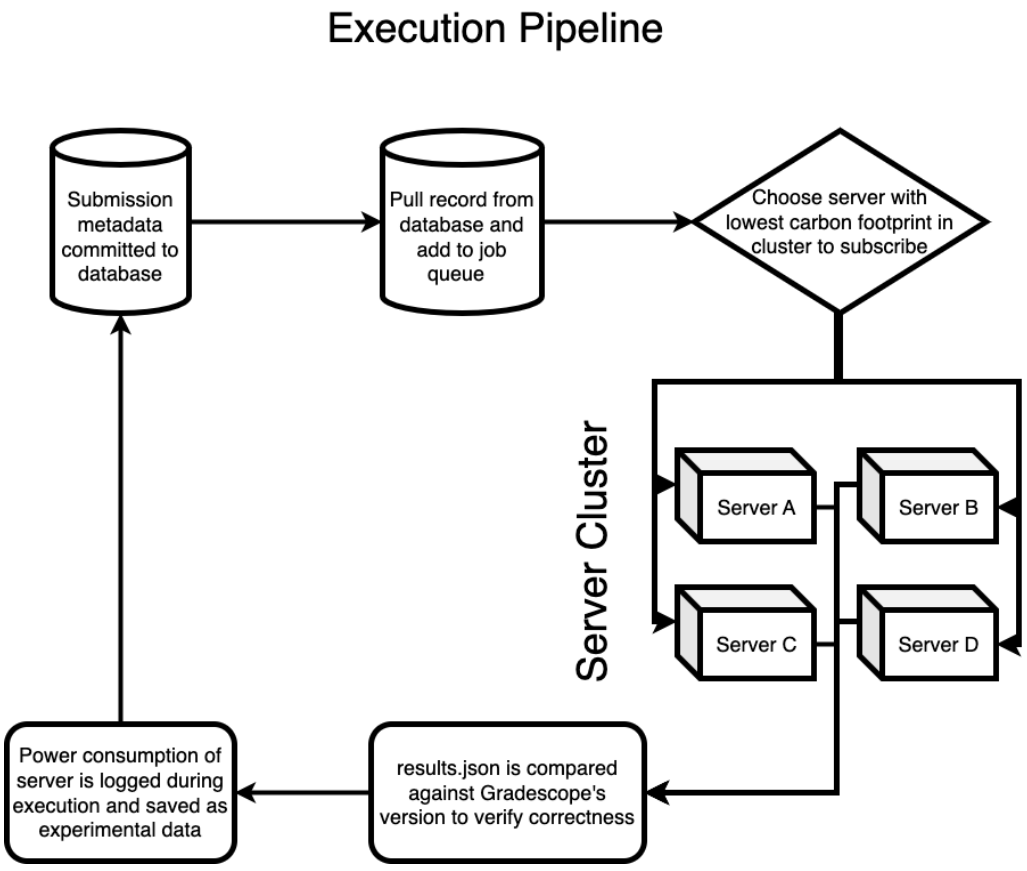


Figure 2: Execution Pipeline Diagram (v1)

### 2.2 Research Objectives

The initial objective was to configure MQTT on two internal UC San Diego servers, ensuring seamless transmission of a data "blob", essentially a composite of a zip file and the requisite metadata. With this foundation in place, the subsequent aim was to integrate this setup with Gradescope's run_autograder script. By accomplishing this, we could extract relevant files and metadata from the autograder environment. Demonstrating that these files, along with their associated metadata, could be securely and successfully transferred to our UC San Diego server was crucial to maintaining the integrity and functionality of the autograding process.

Having established an effective data transfer mechanism, the next objective centered on replicating Gradescope's autograding environment on UC San Diego's infrastructure. By first locally cloning and building the autograder Docker container on a UC San Diego server, we aimed to create a mirror environment for submissions inbound from Gradescope's backend. Upon successful receipt of submissions, a designated shell script would be triggered, effectively building a Docker container with both the submission and autograder data. To ensure the accuracy of this adaptation, we would compare the results.json output with Gradescope's native version. This comparative analysis would ensure that our replication remained true to the original grading standards and metrics and no submission data was lost during the ingestion process.

The integration of these processes into an efficient ingestion pipeline was the next objective in our design. The plan was to assign a dedicated routing server to operate the Mosquitto broker. This server's primary function would be to direct incoming submissions to the most optimal server option, based initially on a rudimentary decision algorithm. In more technical terms, this involved the introduction of a "status" topic in MQTT. This topic would empower client servers to broadcast certain parameters, initially a priority value, which would eventually evolve to encompass power availability data.

The Postgres database would also live on the same server as the Mosquitto broker to house submission data, ensuring organized storage and easy retrieval. To capture the unpredictability and variance of real-world power availability, we would introduce hard-coded adjustable parameters into the servers. These parameters were envisioned to play an important role in testing the selection and queuing algorithm, ensuring that the system always operates under optimal power conditions. The eventual goal was to reincorporate this power-related data

back into our database following the autograder's operation, facilitating comprehensive power analytics and potential areas for further optimization.

# Chapter 3

# METHODOLOGY

## 3.1 Implementation

### 3.1.1 Ingestion Pipeline

In the early phases of the project, our approach sought to employ command-line piping to insert records as zip files arrived via the mosquitto_sub stream. However, challenges in reliability necessitated the creation of a custom ingestion script. A deliberation emerged: either utilize the Mosquitto C Library with its inherent complexities or adopt a Python or Node based generic MQTT solution, which would simplify database operations. Given the project's demands, the latter was chosen, and a test script was promptly devised. Addressing authentication, the primary threat being unauthorized access to the autograder code, we decided to embed the authentication secret within the run_autograder script, making it inaccessible to students. To further enhance security, individual password files were considered for instructors, to be uploaded to Gradescope and securely transmitted via MQTT. As the project advanced, the focus shifted to refining the ingestion pipeline and bolstering security mechanisms.

In our testing, we determined with certainty that the method employed to publish in the run_autograder using mosquitto_pub would neither hang nor timeout due to any delivery failures. Our observations highlighted that this mechanism remained largely oblivious to whether the ingestion pipeline had successfully received the payload. This behavior appeared intentional, stemming from its direct interfacing with the broker rather than the client. While mosquitto_pub does offer repeated delivery policies for enhanced reliability, it always concludes operations gracefully following message transmission. Nonetheless, to address potential concerns about

message receipt acknowledgment, we considered introducing another topic for confirmation or handshake between Gradescope and the client. Though this mechanism could potentially ensure that the ingestion pipeline consistently receives payloads, it became clear that such rigorous confirmation wasn't a pressing concern.

Next, we established a connection to the Mosquitto broker using Python's paho-mqtt module and demonstrated that we could successfully print incoming messages to the command line. At the same time, we set up a database table for submissions and added our first records manually. To ensure the security of our database interactions, we used the SQLAlchemy ORM in Python to guard against SQL injection attacks. We then linked these two processes together in one script to form the basis of our ingestion pipeline. The payload, a directory containing all the relevant submission files, would be generated, compressed, and published to MQTT through the addition of a few lines in Gradescope's run_autograder script. The Python script would then listen for incoming submissions and insert them upon receipt.

Finally, we daemonized the ingestion script to ensure its continual operation in the background, unbound from the user session. This was achieved using 'nohup,' a built-in Unix command that is used to run another command in the background and will keep running even after the user has logged out. In alignment with ensuring the robust health and functionality of the ingestion pipeline, we incorporated NAGIOS, a prevalent and efficient monitoring system. NAGIOS empowered us to actively monitor the pipeline, providing timely alerts and insights into the system's operational status. This precautionary measure further fortified the ingestion pipeline's reliability, protecting it against potential unforeseen discrepancies and operational interruptions.

In a real-world application of the refined ingestion pipeline, it was deployed in the Gradescope environment for CSE 124, an undergraduate course in networked applications at UC San Diego. This deployment captured 134 genuine submissions from students, each securely and systematically stored within the database. This endeavor not only validated the efficacy and reliability of our ingestion pipeline, but also laid a solid foundation for the development of the execution pipeline.

### 3.1.2 Execution Pipeline

Initially, our approach for identifying the server within the cluster having the lowest carbon footprint involved emulating geographic scheduling by regulating servers situated in the UC San Diego basement. This was first achieved by expanding the use of MQTT beyond ingestion to transmit submissions directly to our local cluster servers. Each server in the cluster consistently ran its own version of the execution daemon, individually listening for incoming submissions on their unique topics. Upon the receipt of a submission, the data was stored on the filesystem and uncompressed.

In the subsequent process, a Docker container was dynamically constructed with the directory holding the student's submission mounted as an internal volume. This setup allowed the autograder to operate efficiently, executing the necessary assessment of the submitted work. Post-evaluation, the results were documented on the filesystem, following which the Docker container promptly exited, waiting for the next submission to grade. Multiple Docker containers could be run at the same time, allowing concurrent execution of autograding on each server subject to resource availability. (Figure 3)

Integrating the execution process with our ingestion pipeline presented a dual-faceted challenge. Before execution, submissions needed to be extracted from the database and relayed to the scheduler. This procedure was isolated from the scheduler to accommodate our forthcoming scheduling algorithm, anticipated to query an external API. We termed this intermediary component the "dispatcher", providing a link between the database and the scheduler. The dispatcher's role was to retrieve ungraded submissions from the database, marked by a unique numerical status code, and transmit their metadata to the scheduler. Initially, for testing, a basic round-robin scheduling algorithm was employed as a stand-in for the eventual API-based scheduling mechanism. This scheduler took the responsibility of dispatching the submission to the appropriate MQTT channel for the selected node within the cluster.

The post-execution phase demanded a system to efficiently capture the autograding results alongside any relevant experimental data. Utilizing MQTT once again proved beneficial for this process. Additional functionality was integrated into the execution daemon, enabling it to send the resultant output file from the Docker container’s execution to a designated 'results' topic within MQTT. Simultaneously, a new daemon, named the "receiver," was implemented on the database machine. This daemon, actively listening on the results topic, was designed to capture the results files and incorporate them into a separate results table within the database. This dual action effectively updated the status of the submission record, ensuring it would not re-enter the scheduling queue.

Finally, a system was needed to enable the database to monitor the available resources within our GreenGrader infrastructure. In response, we established a mechanism where a worker node could register to GreenGrader. Upon initiation, the worker would invoke a registration script, transmitting a message via MQTT to the receiver, thereby declaring its intention to

allocate resources. Along with this declaration, the worker transmitted specific system specifications, including CPU model name, clock rate, number of sockets and cores, as well as cache and memory size. These specifications were systematically archived in a new table, designed to keep a record of all available workers. This comprehensive assemblage of components collectively constituted the refined second version of our execution pipeline, marking a significant milestone in the realization of the GreenGrader project. This structure assured an organized and streamlined process, ensuring optimal utilization and management of resources within the system.

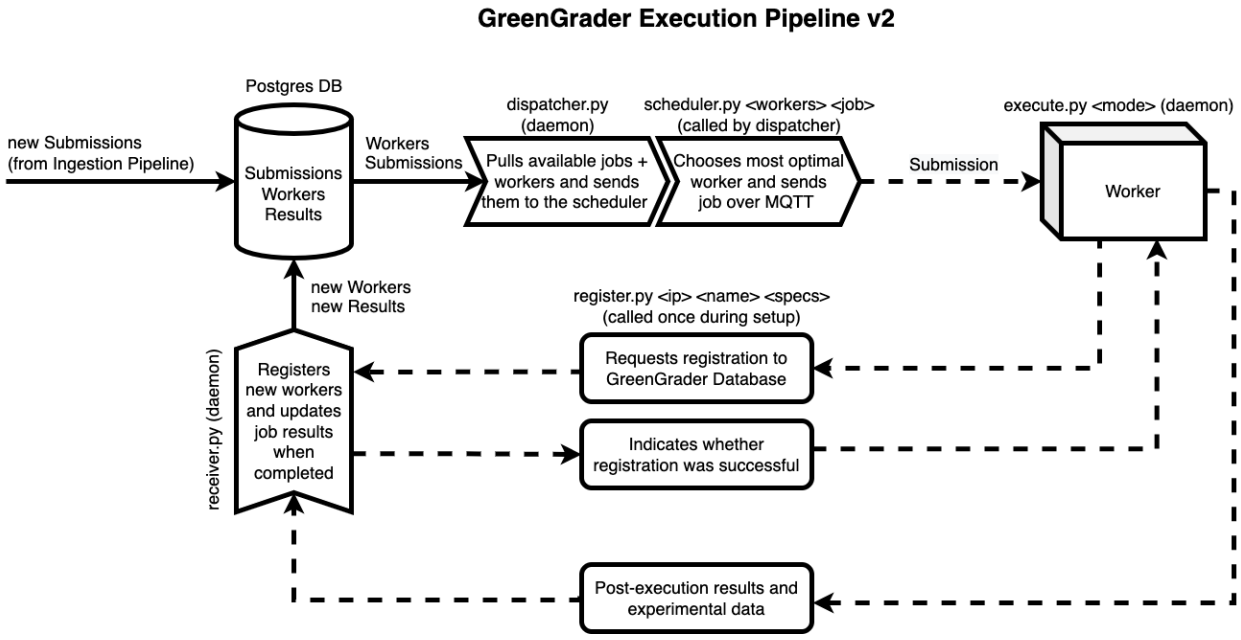


Figure 3: Execution Pipeline Diagram (v2)

### 3.2 National Research Platform (NRP) Integration

#### 3.2.1 Migrating from Docker to Kubernetes

A major turning point in the project was marked by our discovery of the National Research Platform. NRP is a collaborative endeavor that involves over 50 institutions, spearheaded by researchers and cyberinfrastructure experts from UC San Diego. The initiative receives partial funding and awards from the National Science Foundation. One of the primary offerings of NRP is Nautilus, a HyperCluster specifically designed for executing Big Data applications housed in containers. It employs Kubernetes to oversee and scale these containerized applications, working alongside Rook to automate Ceph data services seamlessly. Nautilus provided a new frontier for the project since it afforded us the opportunity to realistically schedule workloads in diverse geographic regions around North America, and even including a few scattered regions in Europe. However, a lot of additional work was going to be required to allow our existing Docker-based execution pipeline to interface with Nautilus' Kubernetes platform.

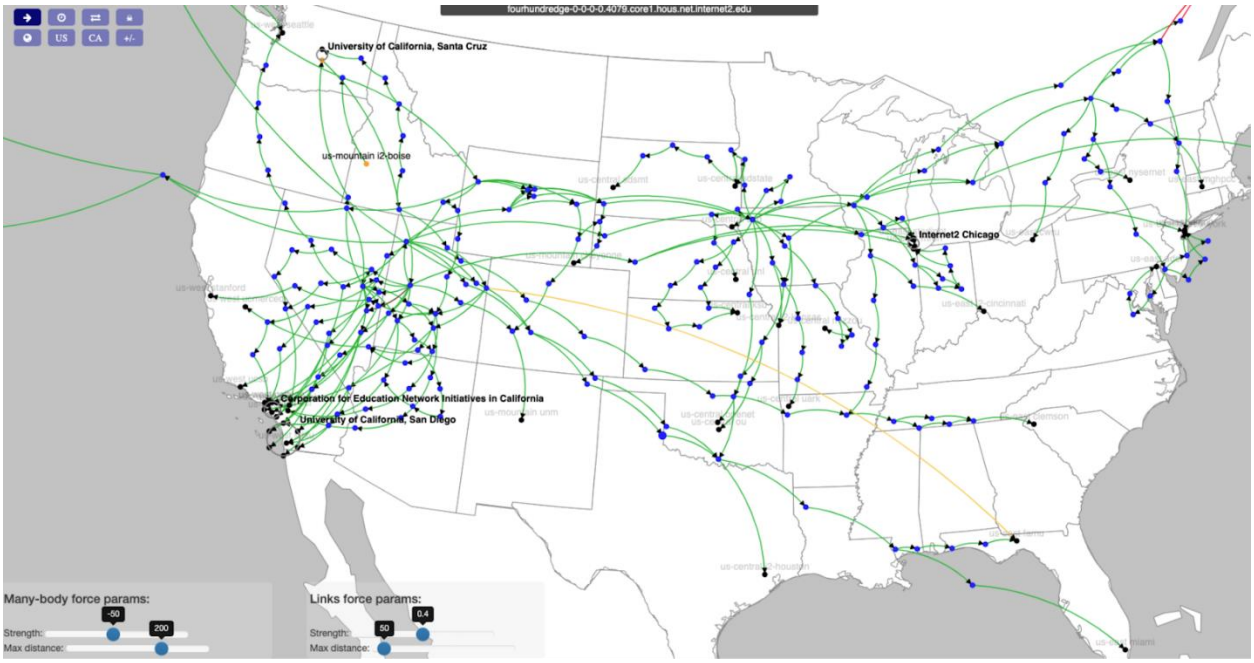


Figure 4: National Research Platform Topology

The initial vision was to extend the functionality of the existing execution pipeline to run in both “Docker mode” and “Kubernetes mode”, however this endeavor posed substantial challenges. While considerable effort was invested in establishing the environment and tools, attaining seamless operation akin to Docker was a struggle primarily due to major differences in container delivery between Docker and Kubernetes. One significant hurdle was the transition from assignment-based container images to reusable containers that could be pulled from a registry. The prior approach entailed building a unique container for each submission, a method inapplicable in the Kubernetes environment, as it was deemed too slow and resource intensive. The introduction of a bind mount for the submission directory, allowing seamless pass-through of submission data into the container, was considered to allow the same image to be reused for multiple submissions. However, this alternative proved incompatible with Kubernetes which prefers Persistent Volume Claims, (PVCs) functioning as abstract storage requests.

Further complicating the task was the issue of container registries. A looming question was how to efficiently transfer Docker images to Kubernetes post-construction. Kubernetes' lack of support for loading images from local context, as previously done, exacerbated the problem. Although a conventional option entails utilizing a container registry that Kubernetes can access, we were hesitant due to concerns about publishing each assignment's container to a public registry like Docker Hub. Potential solutions included employing a private hosted registry like Google Container Registry or establishing our own local registry. While the former offered scalability, it did incur notable expenses; the latter, despite its feasibility, introduces substantial complexity to the project.

Tackling these challenges one by one, we first successfully established a private container registry under the c3lab namespace, providing a secure and efficient repository for our container images. Although pushing to this registry proved relatively straightforward, it was not without its time costs, averaging around seven minutes for a 600MB container. The task of pulling, while operational, unveiled its own set of complexities, demanding a meticulous approach. The worker now necessitated a custom Kubernetes config file for a user in the c3lab namespace, a deploy or personal access token allowing permissions to read and write to the registry, and a shared secret. Despite these additional requirements, the centralized nature of this solution overshadows the alternative of a locally-hosted registry, streamlining the container management process in the larger perspective.

Next, we turned our attention back to the ingestion pipeline and implemented a novel mechanism for automated container building and pushing. The addition of a new table in the database allowed us to maintain a record of assignments and associated containers. As part of this amendment, each submission was now checked to ascertain the existence and version status

of the corresponding assignment in the database. In scenarios where the container is nonexistent or outdated, the system promptly builds and pushes the container to the Nautilus private registry with the corresponding version. This enhancement not only bolstered the efficiency of the ingestion pipeline but also ensured the continuous availability of the latest container versions, optimizing the autograding process.

Venturing into the domain of volume mounts and PVCs in Kubernetes, the journey continued to uncover new challenges. Despite successfully running the autograder container in a Kubernetes pod, the mission of effectively channeling input to the container and procuring output remained elusive. Trials with a PVC had been met with mixed results, and alternatives involving mounting the host directory into the container, mirroring the bind mount approach in Docker, also failed to yield success, with containers refusing to start for undisclosed reasons. However, after weeks of struggling to find the best solution to this problem, we finally were able to harness Nautilus' Ceph storage service to accomplish this goal.

In the final iteration of the updated execution pipeline, all the following operations were combined into one master podspec file. Upon the receipt of a submission, the system utilizes rclone to upload it to a Nautilus bucket via S3. Kubernetes then deploys an initContainer to execute rclone sync, ensuring the submission directory is accurately mounted as a volume into the autograder container. Following successful mounting of the directory, the necessary assignment image is pulled from the container registry and initiates the running of the container, marking the commencement of the autograding process. Concurrently, a sidecar container is deployed that waits for the completion of the autograder container, after which it copies the results into the volume mount. As the autograder container completes its task and exits, the sidecar container waits for the appearance of the results directory in the volume mount. Once the

results materialize, the sidecar container executes rclone sync to upload the directory to the Ceph cloud storage, safeguarding the information and ensuring its availability for further analysis and reference. After the comprehensive completion of the pod operations, the results directory stands ready for download from any machine connected to NRP.

Armed with the capability to operate on NRP, our focus pivoted towards the task of unifying the dispersed components of our system. This entailed rigorous end-to-end testing, spanning from the initial stages of ingestion to the simultaneous execution across multiple locations. Despite successful individual testing in limited scenarios, the amalgamation of these isolated components into a harmonious, seamless operational workflow still posed a challenging venture. Furthermore, we still had one remaining hurdle: integration of our refined system with the carbon-aware scheduler API. This final collaboration aligned with the ultimate objective: to analyze the efficacy of carbon-aware scheduling using GreenGrader as a benchmark.

### 3.3 Carbon-Aware Scheduler (CAS) Integration

The scheduling platform proposed by fellow UC San Diego researcher Yibo Guo presents a comprehensive geographically distributed cloud scheduling system that prioritizes low-carbon power utilization. At its core, the platform deploys a two-level scheduling structure. The first level adjusts the resource footprint across various regions in alignment with their carbon cleanness and current utilization. It focuses on prioritizing regions abundant in low-carbon power when the workload is substantial and minimizing the resource footprint in high-carbon areas when the workloads are fewer. This operates at a frequency, for instance, every 15 minutes. The second level, the job scheduler, operates in real-time. It efficiently allocates individual jobs to optimal locations considering the available resources, carbon cleanness from crawled energy

data, and the migration cost based on data size and WAN bandwidth. The intent is to sidestep the migration of high-cost jobs, preserving carbon savings.

The scheduler essentially maintains a balance between carbon savings and migration costs, ensuring that the energy consumption overhead of moving a workload is substantially outweighed by the carbon savings from such a movement. This balance is managed using a defined migration cost index, calculated as the ratio of a job's predicted compute energy usage to its predicted data size. This metric serves as a guide for making informed and eco-efficient migration decisions among a large set of jobs, ensuring the optimization of both energy usage and carbon footprint reduction.

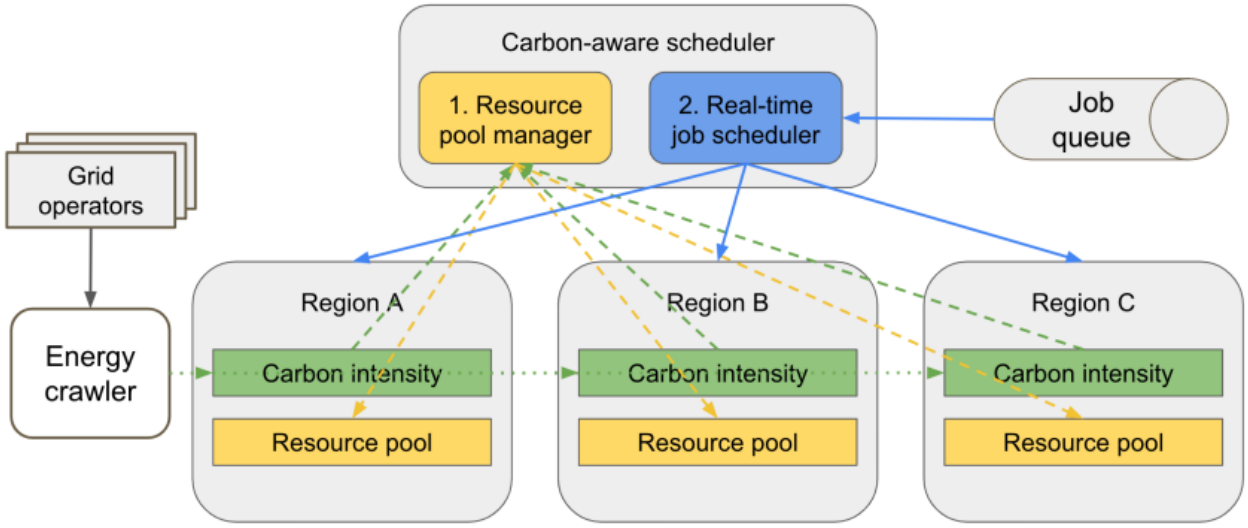


Figure 5: Carbon-Aware Scheduler diagram

The REST API of the scheduler accepts a JSON object, encompassing parameters such as runtime, input/output size, a set of candidate regions, the chosen carbon data source, and desired system resources. This API processes the provided information and responds with a selected

optimal region, alongside a score and carbon emitted values for each region. These critical carbon emission values provided us with valuable metrics for evaluating the GreenGrader project's effectiveness in forthcoming sections.

In harmonizing this with GreenGrader's existing infrastructure, the process unfolds as follows. Initially, the dispatcher formulates an HTTP request to a REST API endpoint, including all submission workload parameters. This step is instrumental in ensuring that the scheduler has all the necessary information to make an informed decision. Following the reception of the optimal region data from the CAS, the construction of a Kubernetes podspec for the specific submission is initiated. The optimal region data is added to the podspec as the workload node "affinity," which is Kubernetes' mechanism for node or group selection within a cluster where the pod should execute. This integration allows for the scheduling and execution of the pod, while the results, upon completion, are downloaded back to the server and committed to the database for further analysis and review.

Nevertheless, this integration process was also not devoid of hurdles. A significant challenge arose with the availability of certain nodes on the NRP network. Occasionally we found that certain selected optimal nodes were marked as "tainted" and refused to execute submissions. These nodes, supposedly restricted in their capacity to accept autograding workloads, posed an obstacle to the smooth operation of the system. Initially, we tried to construct a ranked hierarchy within the node affinity, so that the workload would transition to the next most optimal region in terms of carbon intensity should the initially chosen node be unavailable. However, the end workload location would still deviate from our selected node in the hierarchy. After surveying over 300 nodes in NRP, we found 53 to be completely untainted. Once we assembled this list, we built a map of regions to untainted nodes and passed the

untainted node to the affinity instead. If no untainted node was found within the optimal region returned by the API, the next optimal region with an untainted node would be selected, thus guaranteeing that every workload would be scheduled on the most optimal untainted node available.

The development of the GreenGrader project marks a journey from idea to execution, punctuated by both novelty and challenge. Initial efforts were devoted to building a solid infrastructure, aiming to utilize low-carbon power across various geographical locations. The scope of what we were able to accomplish grew substantially with the discovery of NRP. Its introduction to the project presented us with a whole new set of hurdles to overcome as we rebuilt a lot of components to interface with Kubernetes, a technology that was new to us. The integration with Yibo Guo's carbon-aware scheduling platform reflected the final key development in the project's evolution. The completion of this integration set the stage for the evaluation phase, where the effectiveness of GreenGrader in reducing carbon emissions was assessed.

Chapter 4

RESULTS

## 4.1 Submission Ingestion

Our week-long experimental deployment of the ingestion pipeline in CSE 124 captured 134 submissions from students. The submissions were all for one assignment, titled “Project 4”, in which students had to implement a distributed block storage service in Go. The assignment was then evaluated in the Go environment by testing various operations with the student’s service and ensuring loads were balanced correctly using consistent hashing. Students had the option to upload their submission directly into Gradescope or to submit a link to a GitHub repository hosting their code, and approximately 60% of students preferred to use GitHub as the method of submission.

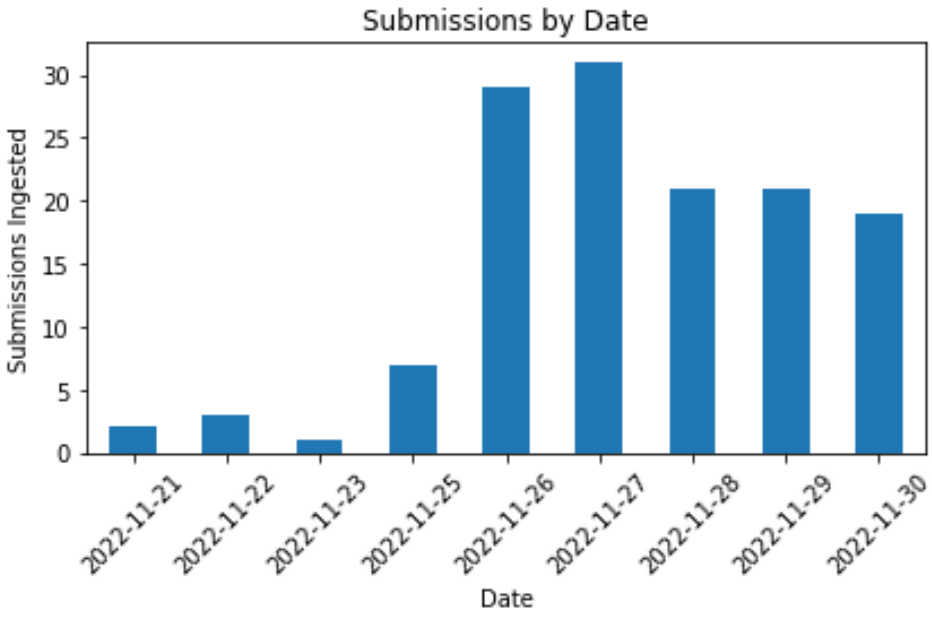


Figure 6: Submissions by Date

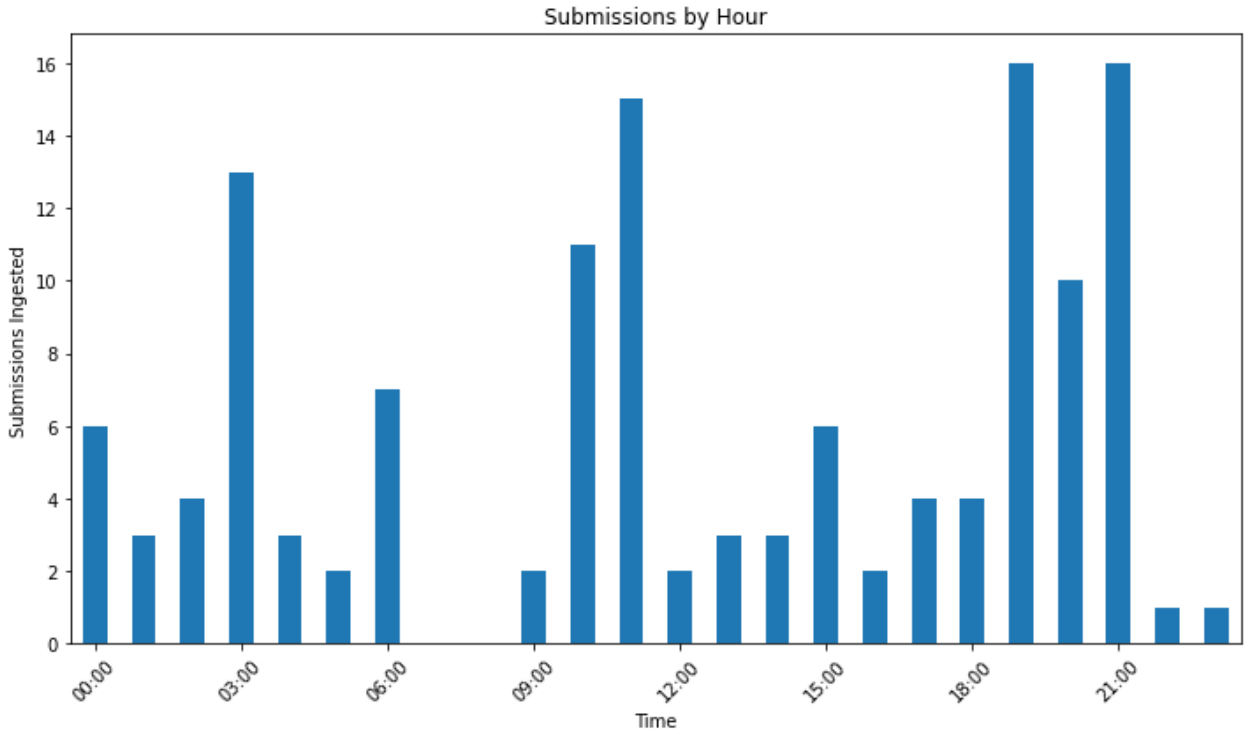


Figure 7: Submissions by Hour

We expected to see more submissions closer to the deadline as students will generally try to use as much time as possible to complete their assignment. Surprisingly, many students chose to submit their assignment far in advance of the November 30th deadline. This may have been due to the fact that a lot of students will use the autograder to verify the correctness of their solution while they are working on it. Furthermore, we saw the highest frequency of submissions occur between the hours of 6-9pm. Overall, submission times seemed to trend towards late morning or mid evening with some scattered submissions in the very early morning. Due to our small sample size, it's difficult to draw a lot of meaningful conclusions from this data as it is only representative of one assignment in one course with a small number of students. However, it does paint a rough picture of autograding workload scheduling with a high degree of variance

depending on the time of day. Generally speaking, more submissions occurred at night, which has consequences for carbon efficiency, as we will examine in the next section.

### 4.2 Real-Time Execution

After running all of the ingested submissions on our GreenGrader infrastructure, we found that the mean runtime for each autograding container was 23.352 seconds with a standard deviation of 2.321 seconds. This was measured on NRP's servers as the time from building the container to the results.json file being generated. The mean turnaround time, measured on GreenGrader's servers as the time from querying the CAS to downloading the results.json file from S3, was a significantly longer 73.427 seconds with a standard deviation of 36.465 seconds. This higher runtime and variance are likely explained by the networking delays that occur when working with the CAS, NRP nodes, and Ceph storage.

Table 1: Summary of Execution Time Data

| | Mean (seconds) | Standard Deviation (seconds) |
|---|---|---|
| Container Runtime | 23.352 | 2.321 |
| Turnaround Time | 73.427 | 36.465 |

Sometimes the NRP nodes would immediately pick up a submission and start executing it; other times, it would get stuck in an initialization state for upwards of 30 seconds, presumably because of high traffic on that node. This highlights a weakness in using NRP's infrastructure: we have no way of knowing what other workloads are being run on a particular node and how

that might affect our throughput. Future iterations of the scheduling algorithm should consider this along with carbon intensity.

### 4.3 Simulated Carbon Emissions

Using the real-time data collected in 5.2, we simulated running GreenGrader in a few different contexts. To simulate execution, we fed each submission's start time (corresponding to the time that the student submitted the assignment) and recorded runtime into the CAS, which uses historical carbon data provided by Green Software Foundation's Carbon-Aware SDK [19] to provide carbon emissions from computation in kgCO2. Note that this simulation assumes unfettered access to all NRP nodes, which is not the case in practice; the data produced in 5.2 is from operating on the subset of untainted nodes.

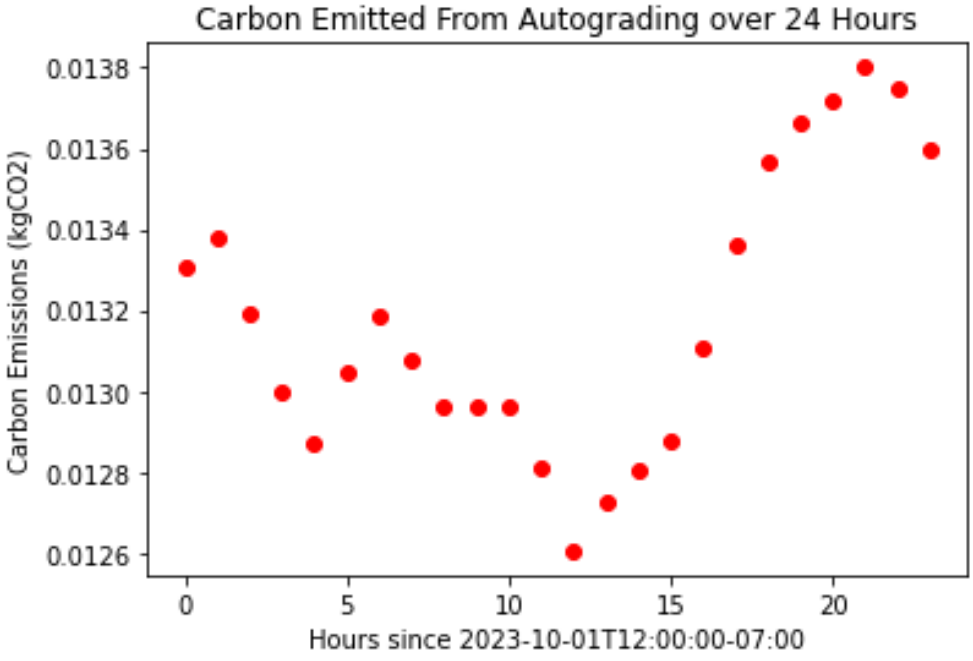


Figure 8: Carbon Emitted From Autograding over 24 hours

We did some temporal analysis to observe how carbon emissions vary over a 24-hour period. We simulated the execution of a single submission once every hour and plotted the carbon emissions in the figure above. As seen in this graph, there is a lot of variation in emissions based on when the submission is graded. Notably, one of the most optimal time periods to run the workloads is from 10pm-12am, which is also a period with higher likelihood of submissions due to deadline proximity.

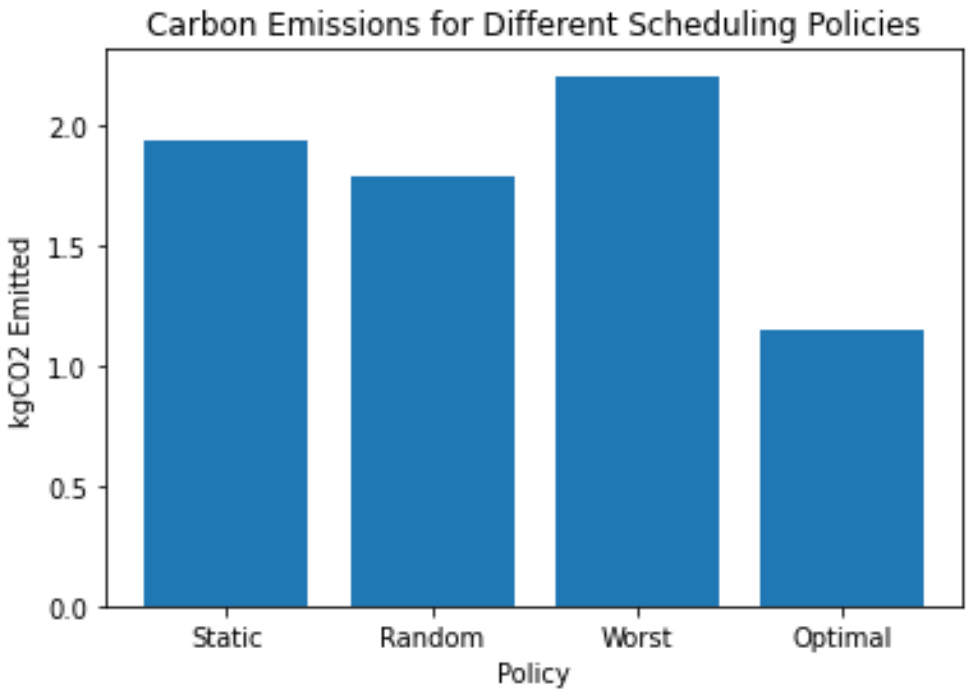


Figure 9: Carbon Emissions for Different Scheduling Policies

Finally, we simulated the execution of all submissions at their respective start times using 4 different scheduling policies. For static scheduling, we simulated scheduling on 'us-west.seattle' every run. We picked this region specifically as the Pacific Northwest is known to be one of the cheapest AWS regions [20]. Furthermore, we also simulated scheduling in random regions, as well as the region with the highest and lowest carbon emissions. (Worst and Optimal,

respectively) We found that GreenGrader's optimal scheduling policy resulted in a 40.91% reduction in carbon emissions from the static policy, a 35.76% reduction from the random policy, and a 48.03% reduction from the worst policy.

# CONCLUSION

This thesis presented the development and evaluation of GreenGrader, a carbon-aware distributed autograder system. The overarching goal was to design and implement an autograding framework that could meaningfully contribute to reductions in carbon emissions from computational workloads. Through the integration of energy-efficient computing, carbon-aware scheduling, and geographic workload distribution, this project sought to align technological advancement with ecological responsibility.

The objectives outlined at the onset revolved around constructing an ingestion pipeline to securely transfer submissions from Gradescope to GreenGrader, developing an execution pipeline to evaluate submissions using Docker containers across a distributed cluster, integrating GreenGrader with a carbon-aware scheduler to enable geographic workload shifting, and analyzing the system's efficacy in reducing carbon emissions. The ingestion and execution pipelines were implemented in phases, gradually enhancing reliability, efficiency, and compatibility with new environments like Kubernetes. The integration with the carbon-aware scheduler API enabled GreenGrader to shift autograding workloads to optimal locations based on real-time carbon intensity data.

The results highlight GreenGrader's ability to meaningfully reduce carbon emissions compared to other scheduling policies like random selection or static geographic placement. When simulated across 134 real student submissions, GreenGrader lowered emissions by 40.91% compared to a static policy. This affirms the potential of carbon-aware scheduling in distributed autograding systems.

While substantial progress was made in realizing GreenGrader's vision, ample opportunities exist to advance and refine the system. On the data collection front, deploying GreenGrader across multiple courses and assignments would provide more robust insights into submission patterns and autograding workloads. Dynamic autoscaling of resources based on submission volume could further optimize efficiency. Exploring alternative virtualization technologies like serverless computing may unlock additional benefits.

In closing, GreenGrader represents an important step in aligning distributed computing with environmental stewardship. Its development required navigating the intersection of education, technology, and sustainability. As society pushes towards a low-carbon future, systems like GreenGrader will grow increasingly vital. With diligence and collective will, the computing community can transform these tools into beacons of ecological promise.

# REFERENCES


[1] Shehabi, Arman, Sarah Josephine Smith, Dale A. Sartor, Richard E. Brown, Magnus Herrlin, Jonathan G. Koomey, Eric R. Masanet, Nathaniel Horner, Inês Lima Azevedo, and William Lintner. "United States Data Center Energy Usage Report." Lawrence Berkeley National Laboratory, June 2016.

[2] Souza, Abel, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. "Ecovisor: A Virtual Energy System for Carbon-Efficient Applications." Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, vol. 2, January 2023, pp. 252-265.

[3] Hanafy, Walid A., Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. "CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency." Proc. ACM Meas. Anal. Comput. Syst., vol. 7, no. 3, article 57, December 2023.

[4] Lindberg, Julia, Bernard C. Lesieutre, and Line A. Roald. "Using Geographic Load Shifting to Reduce Carbon Emissions." 22nd Power Systems Computation Conference, 2022.

[5] Shekhawat, Virendra Singh, Avinash Gautam and Ashish Thakrar, "Datacenter Workload Classification and Characterization: An Empirical Approach," 2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS), Rupnagar, India, 2018, pp. 1-7.

[6] Cisco. "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper." 2018.

[7] Basu, Saptarshi. "A Study on Selection of Data Center Locations." International Journal of Innovative Research in Computer and Communication Engineering, vol. 4, no. 8, pp. 14613-14616, August 2016.

[8] Buyya, Rajkumar, Anton Beloglazov, and Jemal Abawajy. "Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges." Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications, July 2010.

[9] Qureshi, Asfandyar, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. "Cutting the Electric Bill for Internet-Scale Systems." ACM SIGCOMM Computer Communication Review, August 2009.

[10] Masanet, Eric, Arman Shehabi, and Jonathan G. Koomey. "Characteristics of Low-Carbon Data Centres." Nature Climate Change, vol. 3, no. 7, pp. 627-630, June 2013.

[11] Agarwal, Anup, Jinghan Sun, Shadi Noghabi, and Srinivasan Iyengar. “Redesigning Data Centers for Renewable Energy.” HotNets '21: Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks, November 2021.

[12] Maji, Diptyaroop, Prashant Shenoy, and Ramesh K. Sitaraman. “CarbonCast: Multi-Day Forecasting of Grid Carbon Intensity.” BuildSys ’22, November 2022.

[13] Acun, Bilge, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. “Carbon Explorer: A Holistic Framework for Designing Carbon Aware Datacenters.” ASPLOS 2023: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, vol. 2, pp. 118-132, January 2023.

[14] Wiesner, Philipp, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. “Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud.” Middleware ‘21L Proceedings of the 22nd International Middleware Conference, pp. 260-272, December 2021.

[15] Lin, Wei-Ting, Guo Chen, and Huaqing Li. "Carbon-Aware Load Balance Control of Data Centers With Renewable Generations." IEEE Transactions on Cloud Computing, vol. 11, no. 2, pp. 1111-1121, February 2022.

[16] Chen, Changbing, Bingsheng He, and Xueyan Tang. "Green-Aware Workload Scheduling in Geographically Distributed Data Centers." 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, December 2012.

[17] Guo, Yibo, and George Porter. “Carbon-Aware Inter-Datacenter Workload Scheduling and Placement.” Poster Session of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23), 2023.

[18] Radovanovic, Ana, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, Saurav Talukdar, Eric Mullen, Kendal Smith, MariEllen Cottman, and Walfredo Cirne. “Carbon-Aware Computing for Datacenters.” arXiv preprint arXiv:2106.11750, 2021.

[19] Green Software Foundation. "Carbon Aware SDK." GitHub, v1.1.0, https://github.com/Green-Software-Foundation/carbon-aware-sdk. Accessed October 2023.

[20] Amazon. “AWS Local Zones pricing.” https://aws.amazon.com/about-aws/global-infrastructure/localzones/pricing/. Accessed October 2023.