Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Reducing Carbon Emissions in Kubernetes Clusters through Temporal and Spatial Workload Management

Marc Moriabadi

Course of Study:	Informatik
Examiner:	Prof. Dr. Ilche Georgievski
Supervisor:	Dr. Kawsar Haghshenas
Commenced:	September 19, 2022
Completed:	March 19, 2023

Abstract

Carbon aware scheduling is a technique used to optimize the allocation of compute resources in a cloud computing system in order to minimize the carbon emissions associated with running those resources. As the demand for computing resources continues to grow, the carbon emissions associated with data centers and other computing infrastructure are becoming an increasingly significant contributor to climate change. In this paper, we present an overview of the state of the art in carbon aware scheduling techniques for cloud computing systems, including both centralized and decentralized approaches. We discuss the advantages and disadvantages of different carbon aware scheduling approaches, and provide insights into the trade-offs that need to be considered when choosing an approach. Finally, we identify key challenges and open problems in the field of carbon aware scheduling for cloud computing systems, and suggest directions for future research.

Kurzfassung

Die kohlenstoffbewusste Planung ist eine Technik zur Optimierung der Zuweisung von Rechenressourcen in einem Cloud-Computing-System, um die mit dem Betrieb dieser Ressourcen verbundenen Kohlenstoffemissionen zu minimieren. Da die Nachfrage nach Rechenressourcen weiter steigt, tragen die mit Rechenzentren und anderen Recheninfrastrukturen verbundenen Kohlenstoffemissionen immer stärker zum Klimawandel bei. In diesem Beitrag geben wir einen Überblick über den aktuellen Stand der Technik bei kohlenstoffbewussten Planungstechniken für Cloud-Computing-Systeme, einschließlich zentralisierter und dezentralisierter Ansätze. Wir erörtern die Vor- und Nachteile der verschiedenen Ansätze zur klimabewussten Planung und geben Einblicke in die Kompromisse, die bei der Auswahl eines Ansatzes zu berücksichtigen sind. Abschließend identifizieren wir die wichtigsten Herausforderungen und offenen Probleme im Bereich der kohlenstoffbewussten Planung für Cloud-Computing-Systeme und schlagen Richtungen für die zukünftige Forschung vor.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Acronyms

API Application Programming Interface. 18

CLI Command-Line Interface. 21

K8s Kubernetes. 17

kWh kilowatt-hour. 17

LME Local Minimum Error. 27

MAE Mean Absolute Error. 27

MAPE Mean Average Percentage Error. 27

REST REpresentational State Transfer. 20

SLA Service Level Agreement. 17

VCC Virtual Capacity Curve. 24

XGBoost eXtreme Gradient Boosting. 39

1 Introduction

Greenhouse gases, such as carbon dioxide (CO_2) are the most significant driver of observed climate change since the mid-20th century. They build up in the atmosphere and increase the global temperature, leading to changes on land and in the oceans. The greenhouse gases persist in the atmosphere for tens and hundred of years after being released, the effects of the climate change have impact on both present and future [Age]. One of the biggest CO_2 emmitent of today is the electricity industry. For example Germany still heavily relies on the use of fossil fuels, such as coal and gas for electricity generation. According to the Fraunhofer ISE only 45, 7% of electricity is generated by renewable energy sources, such as wind, photovoltaik, water and biogas [ISE21]. The total amount of available renewable energy is growing, but the increase is still lower than the increase in energy demand. That means renewable energy sources cannot keep pace with the rising energy demand, so fossil fuel demand is still going up [CNB21]. Another major problem of renewables is the intermittency. That means renewable energy sources cannot always consistently generate energy at all hours of the day [Ene]. Technologies like wind and solar only generate energy when the wind is blowing or the sun is shining, resulting in fluctuating carbon emissions per kilowatt-hour (kWh) of energy used. Producing high capacity lithium-ion batteries could solve this problem. According to a report published by the International Renewable Energy Agency (IRENA) [IRE17], the cost of lithium-ion batteries has fallen significantly in recent years, but they still remain relatively expensive compared to other forms of energy storage. The report notes that the cost of producing lithium-ion batteries is largely dependent on the cost of raw materials, such as lithium, cobalt, and nickel, which are not only expensive and resource-intensive to extract, but also controversial regarding our environment. Therefore, it is more efficient to consume energy when and where it is generated.

Piontek [Pio22] proposed a Kubernetes (K8s) scheduler that delays the execution of workload to hours with less gCO_2eq/kwh grid emissions. However, this approach does not use a realitstic workload prediction and is purely a proof of concept. Furthermore, the proposed carbon emission model is suboptimal and not suitable for predicting carbon emissions of a specific day. James and Schien [JS19] proposed a method that uses carbon intensity data of a distributed cluster to reduce the gCO_2eq/kwh grid emissions. Nonetheless, this method works only for distributed clusters, where the nodes are located in different regions with varying grid emissions.

In this thesis we propose a method to exploit the fluctuation of carbon emissions in the power grid in a K8s cluster, to decrease the total carbon emissions of a K8s cluster. To achieve this, several methods have to be used e.g. *temporal workload shifting*, which describes the delaying of workload, which is marked as non-critical, until carbon emissions of the local grid are less intense (p.2 [RKS+22]). This assumes the implementation of a carbon emission model to predict an optimal time window for scheduling. Additionally, the standard implementation of the K8s scheduler must be extended to allow for this method to work. Furthermore, a workload prediction must be implemented to prevent shifting workload to times with no available resources, which could impact the Service Level Agreement (SLA), which is a set of rules defined in a contract between service provider and client, to ensure a certain quality of service [Phi11].

An alternative method for reducing carbon emissions is to shift workload not in time, but rather in space. This approach requires a distributed cluster with nodes in different regions, each with its own optimal carbon dioxide emissions window. In order to implement this method, access to an Application Programming Interface (API) providing current values for $gCO2eq/kWh$ is necessary in order to determine the greenest node at any given time. This approach can be effective in increasing the carbon efficiency of workloads, even in cases where a high proportion of *critical* jobs, which cannot be shifted in time, are present. However, in practical scenarios, there may be jobs that need to run on a specific node due to considerations such as privacy or reliance on local data that cannot be easily transmitted.

Another method for reducing carbon emissions is consolidation, which involves maximizing the number of idle nodes in the cluster. This can be achieved by scheduling jobs preferentially on nodes that are already running other jobs, thereby increasing the number of idle nodes that can be shut down to save static power consumption. This approach is most effective in clusters with a high number of nodes and an overall low utilization environment. However, it should be noted that consolidation also increases the risk of outages due to a less distributed cluster, which could violate the SLA.

In general we need flexible workloads to effectively reduce the CO_2 emissions of a cluster. Workloads such as batch jobs often tolerate delays, as long as the amount of computation they perform per day is preserved (p.4 [RKS+22]). To realize that, cluster users could provide a deadline along with the workload. Workloads, that are flexible in space, e.g. text to image diffusion models where only the prompt and the trained model are needed, can simply be performed by the greenest node in the cluster.

In this research paper, we propose a method for predicting the future workload and carbon dioxide emissions of a K8s cluster using time series forecasting techniques. We compare and evaluate various approaches to time series forecasting, including statistical modeling and machine learning methods. We present and test a Temporal Workload Scheduler that utilizes the predictions made by these techniques to minimize the carbon emissions of the cluster. Additionally, we present and evaluate a Spatial Workload Shifter that aims to minimize carbon emissions in distributed clusters. Finally, we compare our K8s carbon-aware scheduling methods and highlight their key advantages.

The structure of this thesis is organized as follows:

- 2 Technical Background: This chapter provides an overview of the technologies used in the implementation of the carbon-aware scheduler.
- 3 Related Work: This chapter presents a review of existing work in the field of carbon-aware scheduling and compares it to the methods proposed in this thesis.
- 4 Design and Implementation: This chapter presents the proposed methods for temporal workload shifting, spatial workload shifting, and predicting workload and carbon emissions.
- 5 Testing and Evaluation: This chapter presents the results of the evaluation of the proposed methods.
- 6 Conclusion and Outlook: This chapter concludes the findings of the research and outlines directions for future work in the field of carbon-aware scheduling.

2 Technical Background

In this section, we will provide a technical background on the technologies and concepts that are central to our research on carbon aware scheduling for K8s. We will introduce K8s, a popular open-source container orchestration platform that enables users to deploy and manage containerized applications at scale [Thu23]. We will also discuss the kube-scheduler, a critical component of K8s that is responsible for assigning pods to nodes in the cluster. We will introduce Minikube, a tool that allows users to run a single-node K8s cluster on their local machine, and Docker, a containerization platform that is frequently used in conjunction with K8s. Finally, we will discuss time series forecasting, a statistical technique that can be used to predict the future grid carbon emission, or optimize the scheduling of workloads to minimize carbon emissions.

2.1 Kubernetes

K8s is an open-source container orchestration system that allows users to automate the deployment, scaling, and management of containerized applications. Originally developed by Google, the Cloud Native Computing Foundation now maintains it [Vio20].

In K8s, containers are grouped into units called pods, which can be managed and scaled together. Pods run on nodes, which are physical or virtual machines that are part of a cluster managed by the K8s system. The underlying architecure is displayed in figure 2.1.

K8s provides several key features that make it useful for managing large-scale, distributed applications:

- It provides users with a declarative configuration model, which means that they can specify the desired state of their application. K8s works to make sure the application meets these requirements.
- It offers self-healing capabilities, which means that if a container crashes or becomes unresponsive, K8s can automatically restart it or replace it with a new one.
- It allows users to scale their applications horizontally by increasing or decreasing the number of replicas of a given container. This makes it easy to manage applications that receive a lot of traffic.
- It provides automatic load balancing, which means that traffic is automatically distributed across the containers in a cluster. This helps to ensure that applications remain responsive even under high load conditions.

As a conclusion, K8s is a powerful tool for managing and scaling containerized applications in a distributed environment [Kub22a].

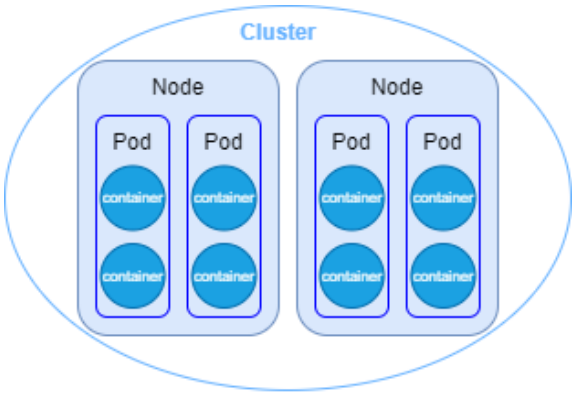


Figure 2.1: Kubernetes architecture

2.2 Scheduler

Scheduling in K8s refers to the process of matching pods to nodes so that the kubelet can run them. The kube-scheduler is a control plane process, that waits for newly created unassigned pods and assigns them to a node. The standard implementation of the kube-scheduler, schedules pods according to their priority. The priority values can range from zero to one billion, higher values are more prioritized. If no priority is assigned to a pod, the pods priority is set to zero. In case pods have the same priority, the scheduler also considers the timestamps and favors pods that have been waiting for a longer time [Kub22b].

Once a pod has been picked the kube-scheduler picks a suitable node for the pod. There are many different ways to influence the node selection process. One possibility is to use node labels, which can be attached manually. Adding labels to nodes, allows pods to target specific nodes, using the nodeSelector. Another selection method is called pod affinity and anti-affinity. Affinity allows pods to be scheduled next to other pods on the same node, depending on the pod label. Anti-affinity works in a similar way, but prevents pods from being scheduled next to pods with a certain label. There can be multiple label requirements and affinities, in this case a node must be found which fulfills all the requirements [Kub22c].

2.2.1 Extending the Scheduler

We see there are some ways to modify the scheduling decision of the kube-scheduler. However, these approaches are limited and do not provide the functionality needed to implement a carbon-aware scheduler. Fortunately, the standard K8s can be easily extended using the *scheduler extender*.

The scheduler extender allows users to define custom rules and policies for the scheduler to follow when making scheduling decisions. The scheduler extender is a simple REpresentational State Transfer (REST) API that allows the scheduler to communicate with the extender by sending HTTP

requests and receiving HTTP responses. The extender can then use this communication to provide the scheduler with additional information about the pods and nodes in the cluster, and to influence the scheduler's decision-making process.

There are three types of plugins that can be used with the K8s scheduler extender: filter, score, and bind. These plugins allow users to customize the scheduling behavior of their clusters by providing the scheduler with additional information and control over the scheduling process. They are capable of being used individually or in combination to achieve a wide range of scheduling behaviors.

- The filter plugin is used to filter the set of nodes that are eligible to host a particular pod. This plugin can be used to exclude nodes that do not meet certain criteria, such as having certain labels or taints. One could use a filter plugin to ensure that pods with certain resource requirements are only scheduled onto nodes that have sufficient resources available.
- The score plugin is used to assign a score to each node in the set of eligible nodes. This score is used by the scheduler to determine which node is the best fit for a specific pod. The score plugin can be used to prioritize certain nodes over others based on various factors, such as the available resources on the node, the workloads already running on the node, or the proximity of the node to other components of the application.
- The bind plugin is used to bind a pod to a node. This plugin is called after the scheduler has selected the best-fit node for the pod, and is responsible for making the necessary changes to the cluster to allocate the resources required by the pod on the chosen node. Additionally, the binding plugin can be used to perform additional validation or to make other customizations to the binding process.

There are other ways of modifying the scheduler, which are not as relevant for our implementation and therefore not discussed [Gui19].

2.3 Minikube

The Minikube tool enables users to run a single-node K8s cluster on their local machine for development and testing purposes. It is useful for developers who are looking to test their applications locally before deploying them to a larger K8s cluster.

Minikube requires a virtualization platform such as VirtualBox or Hyper-V to be installed on your machine and the kubectl Command-Line Interface (CLI) for K8s.

Once these requirements are satisfied, one can use the minikube command to start a local K8s cluster. The cluster can be interacted with using the kubectl CLI.

Minikube also provides some additional features that can be useful for local development and testing. Users can enable a metrics-server addon, which provides a local aggregator of resource usage data. This is necessary for testing our implementation. Furthermore, it allows users to add a dashboard addon, which provides a web-based interface for managing and monitoring the cluster.

In conclusion, Minikube is a useful tool for developers who want to be able to test their applications on a local K8s cluster before deploying them to a production environment. [Kub22c].

2.4 Docker

Docker is a tool that allows users to package applications into lightweight, portable containers. These containers can be run on any platform that supports Docker. A container is a standardized, isolated environment that contains all of the dependencies and libraries required to run an application. This makes it easy to run the application without worrying about conflicts or compatibility issues.

Docker defines the configuration of a container using the concept of a Dockerfile. A Dockerfile contains instructions for building a Docker image, which is a blueprint for a container. The Dockerfile specifies the base image to use for the container, as well as any dependencies or libraries that the application requires, and any other configuration settings that are necessary.

Users can use the docker CLI to build a Docker image based on the instructions in the Dockerfile. The image resulting from this process can then be used to create and run Docker containers.

One advantage of Docker is that it allows users to run applications in a consistent environment, regardless of the host platform. Since the same image can be used on any platform that supports Docker, it is easier to develop, test, and deploy applications.

In addition to this, Docker provides a single container for packaging applications and their dependencies. This simplifies the process of deployment and managing applications in a distributed environment. This is particularly useful in the context of K8s 2.1 [Doc].

2.5 Time Series Forecasting

The task of using a model to predict the future values of a time-dependent series of data points is known as time series prediction. Finance, meteorology, and engineering are some of the fields where this type of prediction is commonly used.

Time series prediction aims to build a model that can take a series of historical data points as input and predict the next value in the series. The model then can be used to make predictions for future values in the series.

Time series prediction can be achieved with several approaches, including statistical modeling, machine learning, and artificial neural networks. The choice of which approach to use depends on the specific data and the goals of the prediction task. Each approach has its own strengths and weaknesses.

For short-term predictions based on known patterns in the data, statistical modeling approaches may be effective, while machine learning approaches may be better suited for making longer-term predictions based on more complex relationships in the data.

Time series prediction is a valuable tool for making predictions about future values in time-dependent series of data points [Shm16].

3 Related Work

Researchers have proposed various techniques for reducing the carbon footprint of datacenters, including energy-efficient scheduling, workload consolidation, and the use of renewable energy sources. In this section, we will review some of the most relevant studies in this area, highlighting their key contributions and limitations. We will focus on approaches that aim to reduce the carbon emissions of datacenters by intelligently scheduling workloads and maximizing the use of energy-efficient resources.

3.1 Spatial Workload Shifting

A Low Carbon Kubernetes Scheduler by Aled James and Daniel Schien [JS19] presents an implementation of a low carbon K8s scheduler, which considers the carbon intensity and local temparature across regions. The paper covers a range of green scheduling approaches, including techniques that consider power consumption, energy cost, carbon emissions, and other factors. The main contributions of this survey are as follows:

- It presents an implementation of the spatial load shifting algorithm.
- It suggests techniques to increase energy efficiency in data centers (e.g. consolidation, virtual capacity planning, load balancing, etc.)
- It presents a carbon emission model that considers the utilization of compute resources as well as the energy consumed during trainsmission of data.

3.2 Temporal Workload Scheduling

CO2 Aware Job Scheduling for Data Centers by Tobias Piontek [Pio22] presents a carbon-aware job scheduler for K8s that considers carbon emissions when allocating resources to jobs. The scheduler is designed to minimize the carbon emissions associated with running a data center while still meeting the performance and reliability requirements of the jobs being scheduled. The main contributions of this thesis are as follows:

- It provides a simulation environment for testing the K8s carbon aware scheduler.
- It presents a carbon-aware job scheduler using the scheduler-extender that considers carbon emissions in addition to other resource constraints when making load shifting decisions.
- It evaluates the performance of the carbon-aware scheduler through simulations, and demonstrates that it is able to significantly reduce carbon emissions while meeting the performance and reliability requirements of the jobs being scheduled.

- It suggests using an advanced Machine Learning Algorithm to improve the precision of the CO_2 prediction, as well as shifting workloads between different locations.

However, this approach is not suitable for real-world scenarios due to the absence of a workload prediction model and a CO_2 prediction model. Additionally, the shifting method is inefficient as it does not consider the amount of shiftable workload to determine the optimal window.

Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud by scientists of the *"Technische Universität Berlin"* [WBS+21] discussed methods for reducing carbon emissions in the cloud using temporal workload shifting.

The main contributions of this survey are as follows:

- It proposes a method to exploit the interruptibility of workloads by carbon-aware schedulers
- It states that the potential for load shifting is generally highest in the early morning hours for countries with a lot of solar power and in the evening hours for countries that throttle their fossil fuel production at night (p.7 [WBS+21]).
- It states that shifting delay-tolerant workloads towards the weekend can result in savings of more than 20% in most regions.
- It found that the greatest savings can be achieved when taking advantage of the interruptibility of workloads (p.12 [WBS+21]).

However, this approach is limited to long running and highly delay-tolerant workload.

Carbon-Aware Computing for Datacenters by Radovanovic et al. [RKS+22] discussed methods for decreasing carbon emissions at Google datacenters, such as shifting execution of flexible workload in time and space, or powering down redundant machines (p.1 [RKS+22]). According to the authors of the journal, the delay decision should be a trade-off between environmental, cost, and modeled performance objectives (p.2 [RKS+22]).

The system's potential comes with the temporal flexibility of a significant fraction of Google's workloads that tolerate delays as long the job is completed within 24 hours. The opportunity to effectively manage the flexible workload lies in the predictability of resource usage and daily consumption within a day-ahead (p.2 [RKS+22]).

The main contributions of this survey are as follows:

- It presents a new method of load shaping with so called Virtual Capacity Curves (VCCs) which are hourly resource usage limits (p.2 [RKS+22]).
- It proposes a total of 5 analytical pipelines necessary for implementing a Carbon-Intelligent Computing system (Carbon fetching-, Power models-, Load forecasting-, Optimization-Service-Level Objective violation detection- pipeline)
- It evaluates the Carbon-Intelligent Computing system performance and demonstrated a power consumption drop of $1 - 2\%$ at times with the highest carbon intensity.

3.3 Other

Assessment of Carbon-Aware Flexibility Measures from Data Centres using Machine Learning by M. Saeed Misaghian et. Al. [MTC+22] presents a study on using machine learning (ML) techniques to predict the energy consumption and server temperature of data centers and integrate these predictions into an optimization framework for power system unit scheduling. The main contributions of this research paper are as follows:

- Comparison of the accuracy of ML models for predicting energy consumption and server temperature to other approaches such as computational fluid dynamics (CFD), resistance-capacitance (RC) network models, and building and energy simulation models.
- Proposal of a framework for integrating ML models into an optimization process to enable a fleet of data centers to participate in a demand response program.
- Exploration of five scenarios for flexibility provision from data centers in terms of their impact on power system operation.
- it proposed series of steps for future work including considering a wider range of system conditions and examining the cost effectiveness and utilization of different strategies.

4 Design and Implementation

The main objective of the implementation is to reduce energy consumption at times of high grid carbon emissions, in order to decrease the overall CO_2 emissions of the K8s cluster across the day. There are two types of workloads, critical and non-critical ones. The critical workloads have to be scheduled immediately and cannot be shifted in time. Non-critical workloads may be delayed up to 24 hours. To achieve a high efficiency in scheduling, the workload must be predictable and consistent.

In this chapter, we describe the different methods used in our implementation of a CO_2 aware scheduler for K8s. In section 4.1 different methods for predicting carbon emissions are compared and evaluated. Section 4.3 presents a method for predicting workload using time series forecasting. In section 4.4 the implementation of the carbon aware scheduler is presented.

4.1 CO2 Prediction

The CO_2 prediction model, is an essential part of the Temporal Workload Scheduler. The precision of the model ultimately determines the efficiency of the scheduling algorithm. A better prediction of the gCO_2eq/kWh values will lead to a better prediction of the optimal scheduling window, which will in turn prevent the workload from being scheduled as early or too late. The data used for the model is kindly provided by [Map], which is the leading resource for 24/7 electricity CO_2 data. We evaluate the most effective strategy for predicting the gCO_2eq/kWh values by comparing the accuracy of different time series prediction models. For evaluating the performance of our prediction models we use following two metrics. The first metric called Mean Average Percentage Error (MAPE) is defined as:

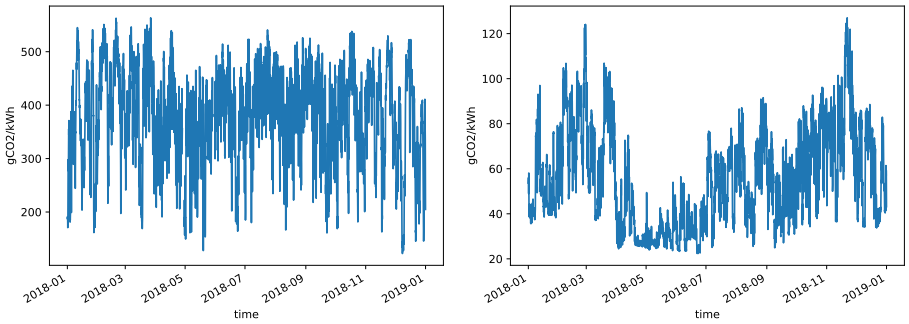
$$MAPE = \frac{100\%}{n} \sum_{t=1}^{T} |\frac{A_t - F_t}{A_t}| \tag{4.1}$$

Where A_t is the observed and F_t is the predicted value. This metric is a good measure for the accuracy of a prediction model and is more interpretable than Mean Absolute Error (MAE). The second metric called Local Minimum Error (LME) is defined as:

$$LME_{(t_0,t_n)} = \frac{1}{n} min(|t_{min(A_t)} - t_{min(F_t)}|, |t_{min(F_t)} - t_{min(A_t)}|), t_0 \leq t \leq t_n, n \in \mathbf{N} \tag{4.2}$$

Where $t_{min(A_t)}$ is the time of the observed minimum and $t_{min(F_t)}$ is the time of the predicted minimum. This is a metric that measures how well our model can predict the local minimum of a certain time interval. This metric is even more important than the MAPE, because the scheduler does not care about the actual predicted value, but only for the local minimum. A prediction model with a low MAPE could still fail to predict the local minimum of the day, resulting in a suboptimal scheduling behavior.

We want our prediction model to be capable of predicting at least 24 hours of CO_2 values, so we can find the optimum within the next day. To ensure that we are picking the overall best performing model, we use different datasets from electricity-map [Map] for comparing the prediction models. We train our model on the entire dataset except the last 24 hours, which we use to determine the predcition error of the model. The first dataset we use are CO_2 signals from Germany of 2018, seen in figure 4.1a. The second dataset we use are CO_2 signals from France of 2018, seen in figure 4.1b.



(a) Germany 2018 **(b)** France 2018

Figure 4.1: CO_2 Signals

4.1.1 Autoregression

The first prediction model we tested is called *ForecasterAutoreg*, which is an autoregressive model from the skforecast library. Autoregression is a method of prediction that uses a linear regression on the previous values, as well as a stochastic (randomly varying) term. The prediction for the first dataset with default settings yielded a MAPE of approximately 10%. The observed daily minimum was at $12:00$, the prediction delivered $23:00$, which results in a high LME of $45,83\%$. The results can be seen in figure 4.2

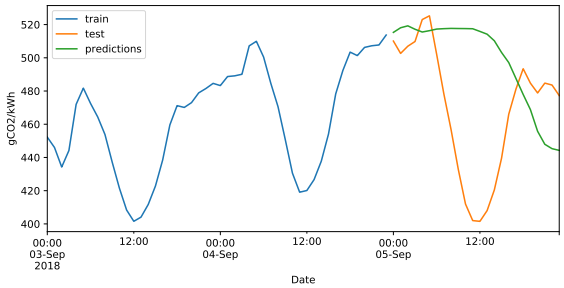


Figure 4.2: Autoregression Germany

Further testing with different model parameters showed that the accuracy of the model could still be improved, as seen in figure 4.3. Using the grid search forecaster, the following parameters were found to be optimal.

- max_depth = 20

- n_estimators = 10
- lags = 400

Training the forecaster with these parameters resulted in a low MAPE of approximately 4%. The observed daily minimum was at 12 : 00 and the prediction was for 14 : 00, resulting in a LME of 8, 33%.

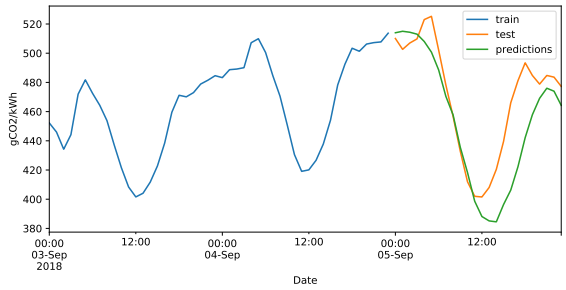


Figure 4.3: Autoregression Germany Optimized

For the second dataset, we initially attempted to use the standard parameters for prediction , the figure can be seen in 4.4. This yielded a low MAPE of approximately 9%. The observed minimum was at 02 : 00, and the prediction was for 03 : 00, resulting in a low LME of 4, 16%.

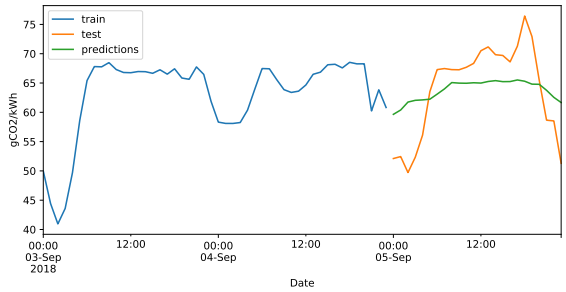


Figure 4.4: Autoregression France

Once again we used the grid search forecaster to find optimal parameters.

- max_depth = 10
- n_estimators = 16
- lags = 400

Training the forecaster with these parameters resulted in a MAPE of 12, 1%. The predicted daily minimum was at 00 : 00 and the observed at 02 : 00 resulting in a LME of 8, 33%. However, the less-than-ideal model parameters in this case were found to give more satisfactory results.

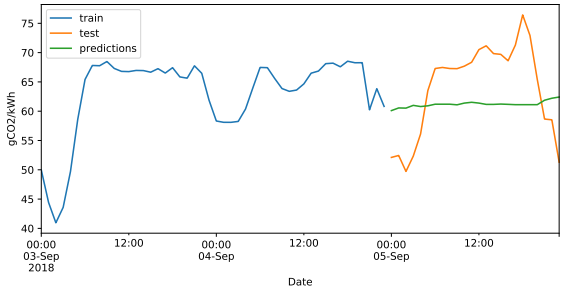


Figure 4.5: Autoregression France Optimized

The optimization for the skforecast models, was achieved by using the grid search forecaster from the scikit library. However, improving the performance of a forecaster can take a considerable amount of time. This is because the model has to be refitted several times with different model parameters. Furthermore, the optimization process requires knowledge about the parameters, since the search grid has to be defined manually. Additionally, the optimization can result in a decrease in the model's precision at predicting a specific day, as seen in figure 4.5.

4.1.2 Prophet

The second approach used for the prediction is called Prophet [Met] and is an open source time series forecasting library created by Meta. The authors explain that the forecasting is based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality. The Prophet model uses the *Stan* algorithm to fit the data, which benefits the forecast speed.

The first training on the first dataset *Germany 2018*, leads to following predictions shown in 4.6. We once again used the last 24 hours of data for testing and trained the model on the remaining data.

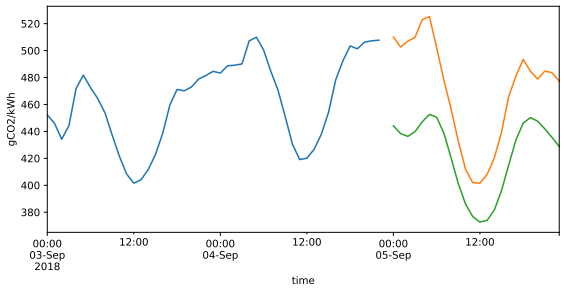


Figure 4.6: Prophet Germany 2018 Predictions

The forecaster performed well not only at predicting the values of a specific day or hour, but also at predicting the behavior overall. The model achieved a low MAPE of approximately 3, 48%, and a LME of 4, 16%. Furthermore, if we look at the predictions in figure 4.7, we can see the following trends:

- The yearly trend of emission is decreasing towards the middle of the year and is again increasing towards the end of the year.

- The lowest emission per week is usually observed on Sunday.
- There is a daily trend of the lowest emission occurring between 10:00 and 14:00.

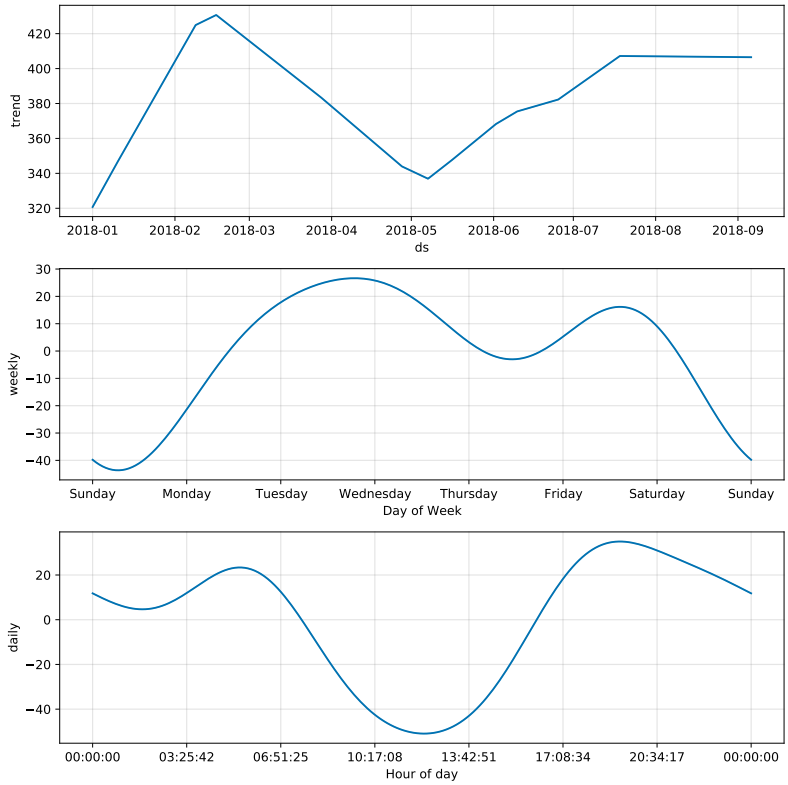


Figure 4.7: Prophet Germany 2018 Trends

The reasons for these trends cannot be explained without guessing. It is safe to assume that energy consumption is lower on Sundays, because it is a rest day in Germany. And reduced energy consumption leads to less burning of fossil fuels, because the amount of renewable energy generated is sufficient. One could also argue that the midday drop is due to the sun being at its highest around that time, resulting in more solar energy.

For the second dataset, the forecaster performed well at predicting the CO_2 emissions, as seen in figure 4.8. The model achieved a low MAPE of approximately $8,58\%$, and a perfect LME of 0%. The trends can be seen in figure 4.9:.

- The yearly trend of emission is decreasing towards the middle of the year and is again increasing towards the end of the year.
- The lowest emission per week is usually observed on Sunday.
- There is a daily trend of the lowest emission occuring around 03:00.

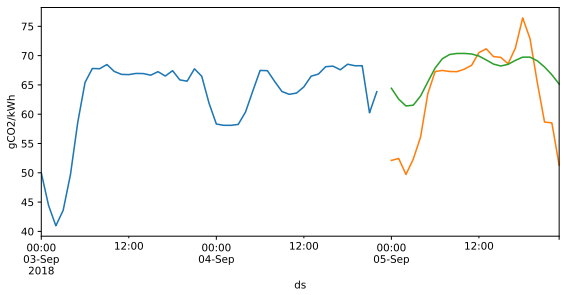


Figure 4.8: Prophet France 2018 Predictions

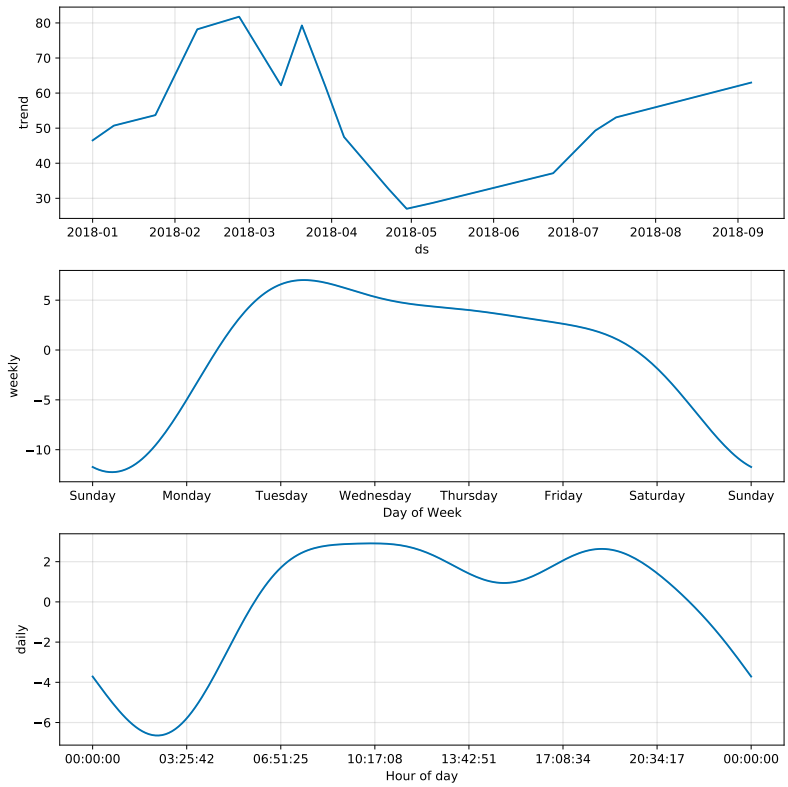


Figure 4.9: Prophet France 2018 Trends

4.1.3 Conclusion

The results of the comparison can be viewed in 4.1 and 4.2. The Prophet model consistently outperforms the AutoregForecaster. Furthermore, it can be fit faster and does not need optimzation compared to the autoregression model.

One potential advantage of the Prophet model is that it is designed for forecasting time series data with trends and seasonality, and can handle missing data and outliers [Met]. This may make it a good choice for datasets that exhibit these characteristics like CO_2 emissions. The Autoregression model is a linear model that makes predictions based on the relationships between the current value

of a time series and its past values [sci]. This may make it a good choice for datasets that are relatively stable and do not have strong trends or seasonality. Therefore, we choose the Prophet model for predicting carbon emissions.

	AutoregForecaster	Prophet
Germany	4%	3, 48%
France	9%	8, 58%

Table 4.1: MAPE

	AutoregForecaster	Prophet
Germany	8.33%	4.16%
France	8.33%	0%

Table 4.2: LME

4.2 Workload Generation

The process of creating a set of data or tasks that are representative of the workload that a system or application is expected to handle is referred to as workload generation. This is typically done to evaluate the performance of the system or application, or to test its ability to handle different types of workloads.

There are several approaches to workload generation, depending on the specific goals and requirements of the evaluation or testing. In some cases, workloads may be generated using synthetic data that is designed to mimic the expected characteristics of the workload. In other cases, workloads may be generated using real-world data that has been collected from a production system.

For this implementation, the workload is generated using real-world data, in particular so-called job traces. Job traces are detailed information about a cluster's workload, including job submission times, job durations, and resource reservations. The information thus obtained can then be used to estimate the hourly workload characteristics of a cluster.

However, the available resources of the target cluster and the real-world cluster usually differ. Therefore, the workload has to be scaled to match the actual cluster's utilization. This is achieved by calculating the relative resource reservation per job rather than the total usage. The relative consumption can then be multiplied by the total resources of the target cluster, resulting in a similar level of utilization.

Additionally, K8s is limited to 110 pods per node. That requires scaling down the number of pods to a manageable amount for single node clusters. In order to compensate for the reduced utilization of the cluster, the resource usage per pod must be scaled up accordingly.

We calculate the utilization per hour as follows:

(4.3)

$$U(h) = \frac{millicores\_per\_job(h) * runtime\_per\_job(h) * total\_jobs(h)}{total\_milli\_cores * 3600} + \frac{static\_millicores}{total\_milli\_cores}$$

The implementation of the workload generation algorithm can be viewed in *„WorkloadGenerator.ipynb"*

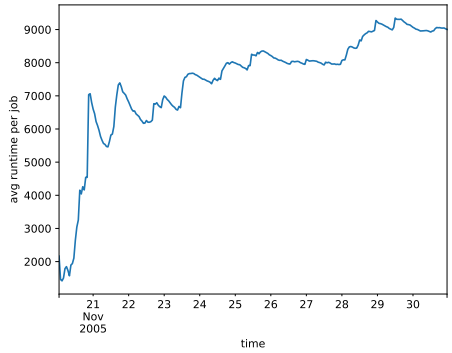
4.3 Workload Prediction

Workload prediction is essential for efficient resource management in server clusters. By analyzing the system logs and identifying patterns and trends in the types of requests received, it is possible to make predictions about future workloads. Statistical modeling techniques, such as regression analysis or time series forecasting, can be used to train models that accurately predict future workloads based on historical data. An accurate workload prediction is crucial for the successful operation of the Temporal Workload Scheduler as it helps to ensure that resources are used efficiently and sustainably.

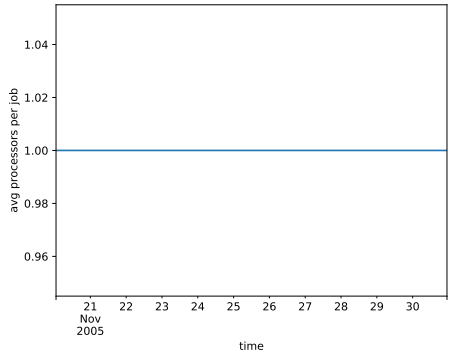
4.3.1 Analyzing System Logs

One approach is to predict the workload of a server cluster by analyzing the system logs. This can provide insight into the types of requests that the server cluster is receiving, and can help to identify patterns and trends that can be used to make predictions about future workloads.

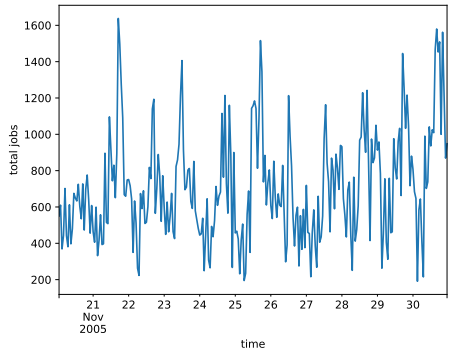
The first dataset we analyze is from a server cluster called GWA-T-11 LCG [05a], which was provided by the e-Science Group of HEP at Imperial College London, and made publicly available by Hui Li through the Parallel Workloads Archive [Tec]. The dataset is a relatively short one with only 11 days, but it was chosen because of its steady and periodic workload characteristic.



(a) Runtime per job



(b) Cores per job

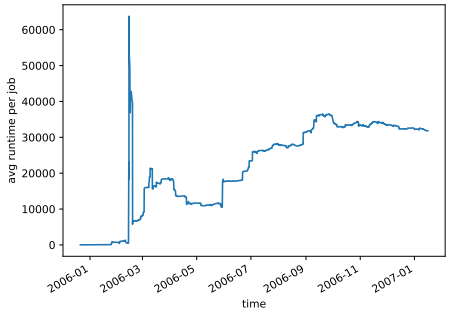


(c) Total jobs

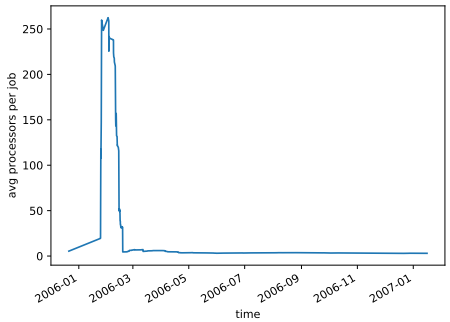
Figure 4.10: Workload components LCG

As the figure 4.10 shows, the runtime and number of cores per job seem to behave predictably. However, the Total Jobs per hour of the underlying server cluster is periodic but noisy. Predicting the Total Jobs per hour assumes using a machine learning or a statistical model.

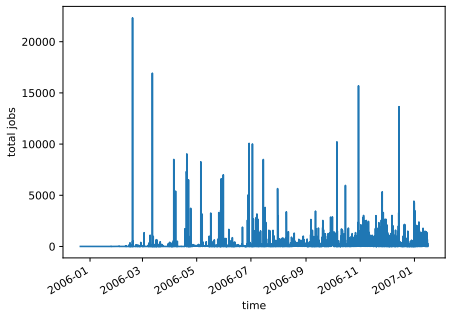
The second dataset we analyze is from a server cluster called GWA-T-10 SHARCNet [05b], which is larger than the GWA-T-11 LCG dataset. However, the SHARCNet dataset is much less steady, aperiodic, and more difficult to predict than the LCG dataset, as seen in figure 4.14.



(a) Runtime per job



(b) Cores per job



(c) Total jobs

Figure 4.11: Workload components SHARCNet

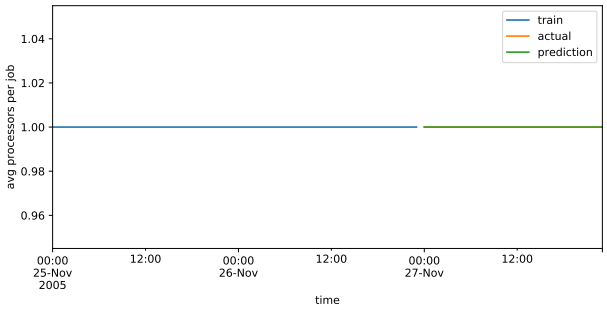
4.3.2 Statistical Modeling

One way to predict the workload of a server cluster is to use statistical modeling techniques, such as regression analysis or time series forecasting. This approach involves using data on the workload of the server cluster over time to train a statistical model that can predict future workloads.

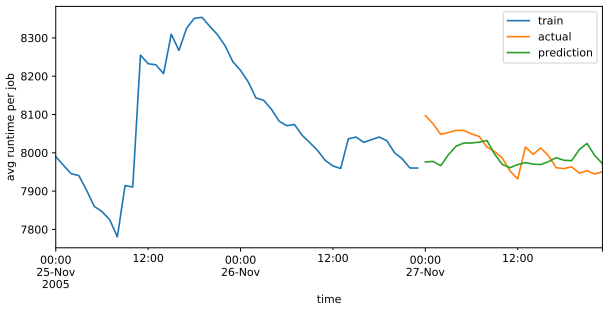
For the time series forecasting models we once again used the AutoregForecaster from the skforecast library and the Prophet model from Meta [Met].

Figure 4.12a shows the cores per job prediction for the LCG dataset using the AutoregForecaster model. It can be observed, that the models prediction is accurate with a MAPE of 0%. This was to be expected, due to the constancy of the historical data.

Figure 4.12b shows the runtime per job prediction for the LCG dataset using the AutoregForecaster model. The model has a high accuracy with a negilible MAPE of approximately 0.4%. This is accurate enough to precisely predict the resource reservation of jobs in the workload.



(a) CPU Prediction



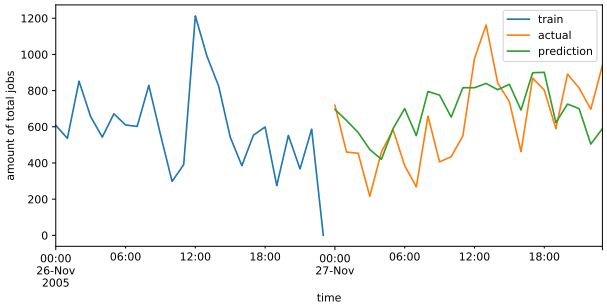
(b) Runtime Prediction

Figure 4.12: AutoregForecaster Prediction LCG

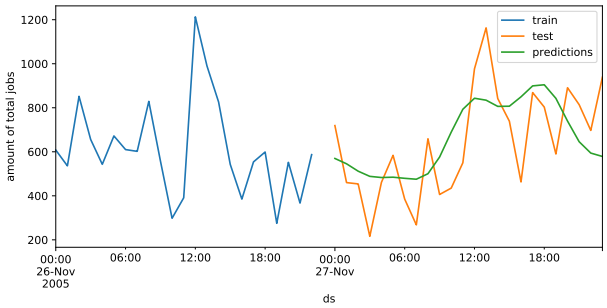
However, the main challenge of predicting workloads is to predict the total amount of jobs at a specific point in time. As stated in 4.3.1 the total jobs per hour of the LCG cluster are periodic but noisy. The figure 4.13a shows a prediction of the total jobs per hour for the LCG dataset using the AutoregForecaster model. The MAPE of prediction is approximately 34.3%, which can be improved.

Using the Prophet model we receive following prediction, seen in figure 4.13b. The MAPE of the prediction is approximately 31.6%, which is slightly better than the AutoregForecaster prediction.

In figure 4.14, we can see the predictions for the SHARCNet dataset. Again, the AutoregForecaster prediction was sufficient for the runtime and cores per job prediction, with a neglible MAPE of $< 0.1\%$. Still, the AutoregForecaster's prediction of the total number of jobs is hihgly inaccurate with a MAPE of 1989.9%, which is unacceptable for predicting workloads. The Prophet model on the other hand managed to predict the total number of jobs with a relatively high accuracy compared to the AutoregForecaster, achieving a MAPE of 90.3%. Nonetheless, both models performed suboptimal due to the noise in the historical data. As stated in subsection 4.3.1 the aperiodic and noisy data is difficult to predict. Therefore, a MAPE of 90.3% may be considered an acceptable prediction error.

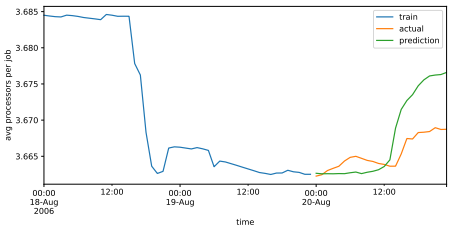


(a) AutoregForecaster

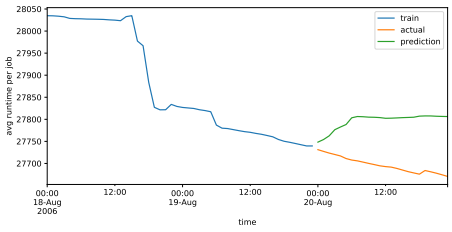


(b) Prophet

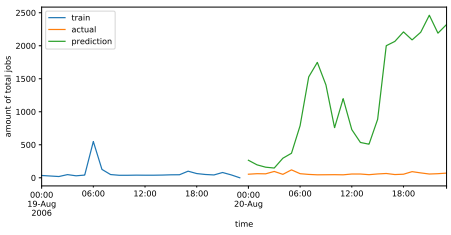
Figure 4.13: Total jobs per hour LCG dataset



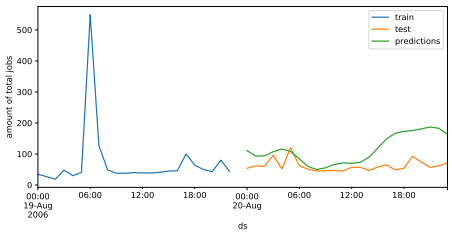
(a) Cores per job prediction AutoregForecaster



(b) Runtime per job prediction AutoregForecaster



(c) Total jobs prediction AutoregForecaster



(d) Total jobs prediction Prophet

Figure 4.14: Workload prediction SHARCNet

4.3.3 Machine learning

Another approach for predicting the workload of a server cluster is to use machine learning algorithms. This involves training a machine learning model on historical workload data, and using the trained model to make predictions about future workloads.

For the machine learning model we are using eXtreme Gradient Boosting (XGBoost), which is a popular and efficient open-source software library for implementing the gradient boosting machine learning algorithm. Gradient boosting is a type of ensemble learning algorithm that combines the predictions of multiple weak models, such as decision trees, to create a strong model.

One of the key features of XGBoost is its ability to handle missing values and imbalanced datasets, which are common in real-world data and especially for our datasets. It also provides a range of hyperparameters that can be fine-tuned to optimize the models performance [XGB].

Unlike statistical models 4.3.2 which use previous values to predict ahead, machine learning works by training the model on features and target value pairs. Therefore, the first step of a machine learning time series prediction is called feature creation. We decided to use the features *'hour'*, *'dayofyear'*, *'month'*, *'dayofweek'*, which can all be derived from the date. The target value respectively is the total amount of jobs at the given date.

It is possible for XGBoost to rank features based on their importance. In figure 4.15, the importance of each feature is shown.

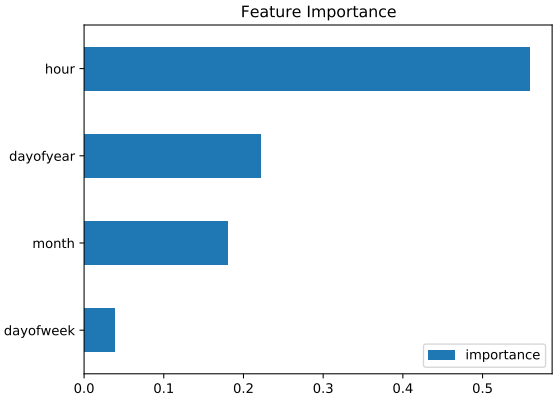
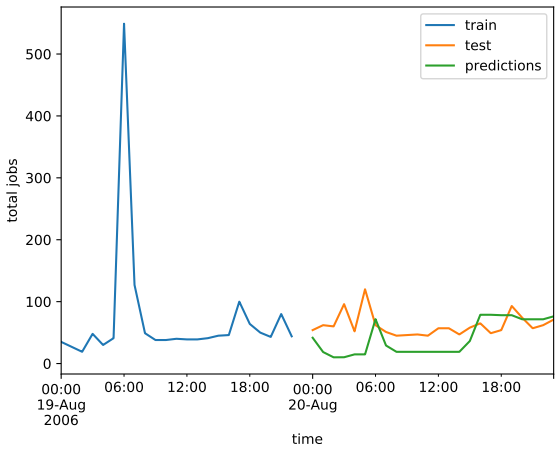


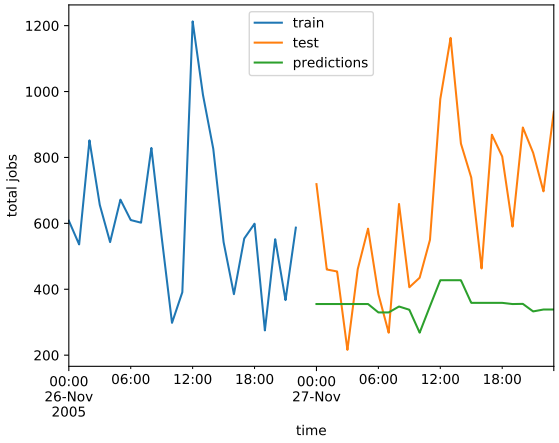
Figure 4.15: Feature Importance

In figure 4.16a the XGBoost prediction of the total amount of jobs for the SHARCNet dataset is shown. The prediction performs much better than the AutoregForecaster and Prophet predictions, which are shown in 4.14. The XGBoost model achieves a MAPE of 47.5%, which is relatively low compared to the statistical models error.

However, regarding the LCG dataset the XGBoost model performs worse than the statistical models with a MAPE of 42.8%, as seen in figure 4.16b. This is because the size of the LCG dataset is much smaller than the SHARCNet dataset. Machine learning algorithms can be very complex, with many different parameters that need to be optimized. The more data an algorithm has to work with, the better it can learn the underlying patterns in the data and find the optimal values for these parameters.



(a) SHARCNet



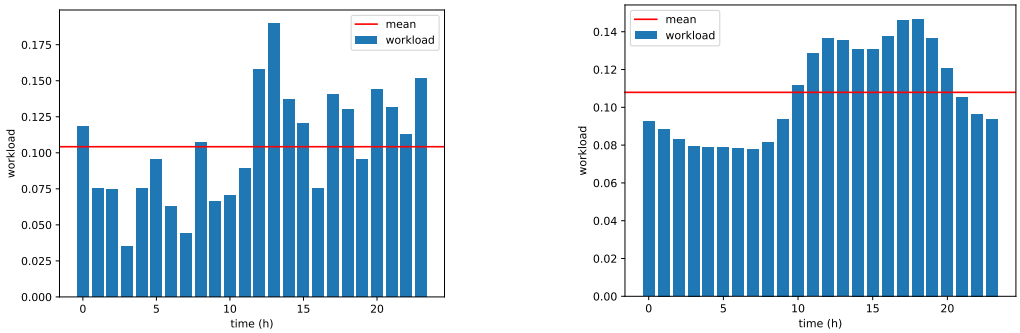
(b) LCG

Figure 4.16: Total jobs per hour XGB prediction

4.3.4 Workload Prediction

After predicting the necessary workload components in the previous subsections, the total utilization for each hour can be predicted using the equation 4.3. The prediction of total workload utilization for each hour is performed using the AutoregForecaster model for runtime estimation and the number of processors, and the Prophet model for the LCG dataset and the XGBoost model for the larger and noisier SHARCNet dataset for the total number of jobs per hour.

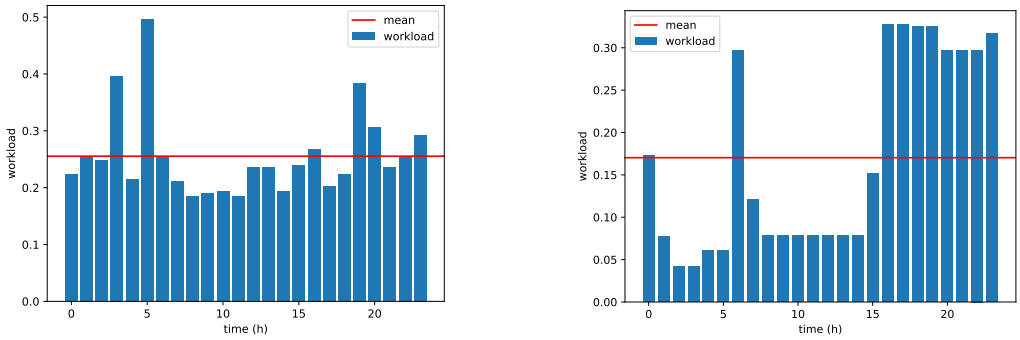
In Figure 4.17, the predicted workload utilization and the actual workload utilization for the LCG dataset are depicted, with a MAPE of 31.3%. The predicted mean utilization is only 3.5% lower than the actual mean utilization.



(a) Actual utilization **(b)** Predicted utilization

Figure 4.17: Workload prediction LCG

In figure 4.18, the predicted and actual workload utilization of the SHARCNet dataset can be seen. The MAPE of the prediction is higher with approximately 47.69% than the LCG prediction, this was to be expected and is due to the inaccurate prediction of the total number of jobs. The predicted mean utilization is 33.3% lower than the actual mean utilization.



(a) Actual utilization **(b)** Predicted utilization

Figure 4.18: Workload prediction SHARCNet

4.4 Temporal Workload Scheduler

The goal of the proposed Temporal Workload Scheduler is to delay non-critical workload to periods of the day with lower grid emissions in order to reduce the total carbon emissions of the cluster. To achieve this, the Workload Scheduler leverages the workload prediction method described in Section 4.3 and the CO_2 prediction method described in Section 4.1.

Carbon emissions can be significantly reduced by shifting workloads to periods of the day when grid emissions are lower. This is especially effective if the cluster is located in an area with significant differences in grid emissions between peak and off-peak periods. Delaying non-critical workload can also help to optimize the utilization of resources in the cluster. By shifting workloads to periods of the day when resources are underutilized, the cluster can operate more efficiently, potentially reducing overall energy consumption.

However there are several challenges to consider when implementing a strategy to delay non-critical workload in order to reduce carbon emissions. As we mentioned in Section 4.3, predicting workload patterns and determining the optimal periods of the day to shift workloads can be complex, especially if the workloads are highly variable or the grid emissions are difficult to predict. Delaying workloads may also increase the overall time required to complete a task, which may not be acceptable for some types of workloads or for users with strict deadlines.

The proposed algorithm aims to minimize carbon emissions by maximizing the utilization of the cluster at optimal hours with lower grid emissions. To achieve this, the algorithm utilizes information about the current non-critical workload, including its CPU reservation, estimated runtime, and total number of jobs in the queue. The runtime is estimated using an autoregression model (4.3.2), while the CPU reservation and total number of jobs can be extracted from the metadata of the pod specification. To prevent non-critical workload from being scheduled at high grid carbon emission hours, the algorthm utilizes the filter plugin, which was mentioned in 2.2.1.

The algorithm begins by sorting the hours of the timeframe in ascending order of CO_2 emission predictions, with the hour of lowest emissions at index 0 and the hour of highest emissions at index n (where n is the length of the timeframe). This step is only performed once at the initialization of the scheduler. It then iterates through the CO2 optimal hours starting at index 0, checking if the optimal hour has already passed in order to prevent shifting workload into the past. The algorithm uses the workload prediction to determine how much of the workload can be shifted towards the optimal hour, taking into account the dynamically determined CPU limit, which uses information such as the workload prediction, static workload and current workload of the cluster. If the optimal hour's CPU limit is reached, the algorithm increments the CO2 optimal hours index and shifts the remaining workload to the next best hour. The next best hour in this context refers to the hour with the next lowest gCO_2eq/kWh value. This process continues until the remaining workload in the queue reached 0. The algorithm eventually returns an array of length n, with values in the range of $[0, 1]$ representing the allocation of CPU to non-critical workload at each hour of the timeframe. This array is then used in the scheduling decision, and limits the cpu reservation for non-critical workloads. Pseudocode of the Temporal Workload Scheduler can be viewed in Algorithm 4.1.

The most important part of the algorithm is to define the optimal CPU limits, in order to maximize utilization and SLA compliance. A high limit can lead to a more utilized cluster, but it can also increase the waiting time for critical workloads. Low limits reduce the probability of a delay in critical workload. But it also limits overall potential savings due to a less utilized cluster at hours with low carbon emission.

One advantage of this algorithm compared to the carbon-aware scheduler proposed by Piontek [Pio22] is that it uses real-time estimations of the workload in the queue, leading to more optimal scheduling windows. However, incorrect workload predictions can have negative impacts on the effectiveness of delaying non-critical workload to reduce carbon emissions in a K8s cluster. If the workload prediction is overestimated, it may result in shifting too much workload to a specific hour of the day, potentially leading to resource contention and insufficient resources in the cluster for critical workloads that need to be scheduled immediately. This can have negative consequences on the performance and reliability of the cluster, potentially affecting the quality of service and SLA for critical workload. On the other hand, if the workload prediction is underestimated, it may result in insufficient utilization of resources during low emission hours, leading to reduced carbon emission savings. Therefore, it is important to accurately predict workload in order to effectively delay non-critical workload and maximize carbon emission savings in a K8s cluster.

Algorithm 4.1 CO_2 Optimal Window

```
H ← hour with minimum predicted CO2 emission in the specified timeframe
queue ← get workload in queue
while queue > 0 do
    limit ← Calculate dynamic limit for H using workload prediction and currently scheduled
jobs
    if H > length of the timeframe then
        optimal_workload[now] ← limit                    // Schedule all workload immediately
    end if
    if H has already past then
        H ← hour with the next minimum predicted CO2 emission
    end if
    if queue fits in H then
        optimal_workload[H] ← workload_prediction[H] + static + shifted_workload[H] + queue
        queue ← 0
    else
        optimal_workload[H] ← limit
        queue ← queue - scheduled workload
    end if
end while
```

Incorrect CO2 predictions can lead to shifting workload towards hours with higher carbon emissions, which can potentially undermine the goal of reducing the carbon footprint of the cluster. This is because the Workload Scheduler relies on accurate CO2 predictions to determine the optimal hours for scheduling non-critical workload. If the CO2 predictions are incorrect, the Workload Scheduler may inadvertently schedule non-critical workload during hours with higher carbon emissions, resulting in a higher overall carbon footprint for the cluster. This could occur, for example, if the CO2 predictions underestimate the actual carbon emissions of a particular hour, leading the Workload Scheduler to schedule non-critical workload during that hour. To mitigate this risk, it is important to ensure that the CO2 predictions used by the Workload Scheduler are as accurate as possible, either through the use of reliable CO2 emission data or by implementing robust prediction algorithms.

4.5 Spatial Workload Shifter

Shifting workload between nodes can be an effective strategy for reducing carbon emissions in a K8s cluster. By distributing workloads across different nodes, it is possible to take advantage of varying carbon emissions patterns throughout the day. For example, if certain nodes are located in areas with lower grid carbon emissions during certain hours of the day, shifting workloads to those nodes during those times can result in significant carbon emission savings. However, it requires the availability of multiple nodes in different locations with significantly varying carbon emissions patterns.

The function $carbon\_score$ 4.2 is a Score plugin for the K8s scheduler extender. It takes in a pod and a list of nodes as input and returns a *HostPriorityList*, which is a list of HostPriority structs mapping scores of the type integer to nodes. The *HostPriorityList* will be used in the Bind step of scheduling, where the node with the highest score will be chosen to host the pod.

The function initializes an empty *HostPriorityList*, here mentioned as score. It then calculates the current hour in UTC+1 time and assigns a low score of 0 to all nodes in the cluster. The function then determines the node with the lowest carbon emission for the current hour and assigns it a high score of 100. This high score difference is necessary to ensure that the prioritized pod is assigned to the node with the lowest carbon emission. Without a significant score difference, the scheduler may consider other factors such as the utilization and distribute the pods across multiple nodes. The function then returns the *HostPriorityList*. The current carbon inensity of a specific location can be retrieved using the API of *electricitymaps.com* [Map].

Algorithm 4.2 Carbon Score Algorithm

```
min ← ∞
greenest ← ∅
score ← ∅
for all node ∈ Nodes do
    score[node] ← 0
    if getCarbonIntensity(node) < min then
        min ← getCurrentCarbonIntensity(node)        // Use API call or prediction
        greenest ← node
    end if
end for
score[greenest] ← 100
```

5 Testing and Evaluation

In this chapter, we will describe the process of testing and implementing the proposed carbon aware scheduling approach for datacenters. We will provide details on the hardware and software requirements for running the tests, as well as the steps involved in setting up and configuring the testing environment. We will also discuss the methodologies and metrics used to evaluate the performance of the approach, and present the results of the tests. Finally, we will discuss the challenges and limitations encountered during the testing and implementation process, and offer insights on how these challenges can be addressed in future work.

5.1 Requirements

In order to test the implementation, a machine with a Windows 10 operating system or higher is required. The Minikube cluster must have at least 4 cores, and virtualization must be enabled in the UEFI settings. Additionally, a virtualization software such as Hyper-V must be installed on the machine. Docker must also be installed, and a Docker Hub account with a repository is required. These requirements must be met in order to properly test the implementation of the proposed carbon aware scheduler.

These are the steps to create a single node testing environment:

1. create a new repository in DockerHub
2. run $minikube\ start\ --\ cpus\ 4$
3. run $minikube\ addons\ enable\ metrics\ -\ server$ (wait until metrics are available, this can be checked with *kubectl top nodes*)
4. run and log in to Docker Desktop
5. run $benchmark\_scripts/workload\_generatorWorkloadGenerator.ipynb$ for workload generation/prediction (further instructions file)
6. run $co2\_prediction/co2prediction.ipynb$ for co2 emission data (further instructions file)
7. change IMAGE=USERNAME/REPOSITORY accordingly in $REPOSITORY/deployScheduler.sh$ $(l.6)$
8. set $target\_time$ in $REPOSITORY/deployScheduler.sh$ accordingly (make sure to set target time at least 5 minutes earlier than the actual start time, because the initialization takes time) and run it in bash
9. set $target\_time$ in $REPOSITORY/benchmark\_scripts/workload\_generator/workloadGenerator.sh$ to your start time and run it in bash

10. set *target_time* in *REPOSITORY/benchmark_scripts/log_data/logData.sh* to your start time and run it in bash

For a multinode cluster environment, *minikube start –cpus 2 –nodes 2* must be used. Additionally the nodes have to be labeled according to their location using the command *kubectl label nodes <nameOfNode> location=<value>*. The value hereby refers to the index of the CO_2 emissions file.

5.2 Methodologies and Metrics

To evaluate the performance of the scheduler, we calculate the total carbon emission savings compared to the standard implementation of the scheduler. However, total carbon emission savings alone do not provide a complete picture, so we also present the relative carbon emission savings, and the achieved SLA compliance.

It is important to note that the carbon emission savings are also influenced by factors such as the utilization of the cluster, the proportion of non-critical workload and the daily optimal hours relative to the time the test is initiated. In a more utilized cluster, the proportion of static energy consumption is reduced, resulting in a higher relative shift of energy consumption to more optimal hours. Furthermore, a higher proportion of non-critical workload increases the amount of flexible workload being scheduled, ultimately increasing the performance of the scheduler. Initiating the test immediately preceding the optimal hours would result in less workload being scheduled and more workload being deployed immediately, thereby potentially impacting the performance of the scheduler. This issue only arises in a 24-hour testing scenario, where the algorithm is limited to scheduling workloads within a fixed 24-hour window. As a result, workload that is deployed at the last hour of the window cannot be scheduled. In a real-world scenario, the algorithm would run continuously, allowing for the scheduling of workloads up to 24 hours into the future.

The intraday fluctuation of CO2 values is another significant factor that can impact the performance of the scheduler. If there are minimal differences in CO2 emissions between the hours, the savings that can be achieved will be minimal or nonexistent. On the other hand, if there are unusual dips in CO2 emissions, the performance of the scheduler may be significantly increased.

It is important to note that there are many factors that can influence the performance of the scheduler beyond the scheduler itself. Therefore, it is not recommended to compare the performance of the scheduler to other research without considering these factors. To accurately compare the performance of different scheduling methods, it is necessary to use similar testing environments.

5.3 Service Level Agreement

A service level agreement (SLA) is a contract between a service provider and a customer that specifies the terms and conditions under which a service is provided. SLAs are commonly used in the context of managed IT services, cloud computing, and other technical services. The purpose of an SLA is to ensure that the service provider meets certain performance standards and commitments, and to provide a clear understanding of the service level that the customer can expect to receive [Phi11].

In our test scenario, the only SLA requirement is the immediate execution of critical workload. The number of SLA violations is measured by counting the number of critical jobs that cannot be deployed due to a lack of available resources. This can be measured using the *kubectl get pods -l realtime=critical –field-selector=status.phase=Pending –namespace=pod-benchmark* command, which retrieves a list of currently pending critical pods.

5.4 Performance Logging

Logging the performance of the K8s cluster is an essential aspect of testing and evaluating the scheduler. To assess the energy consumption of the cluster, detailed information about CPU utilization at any given time is required. This information can be obtained using the *kubectl describe nodes* command. It is important to note that the metrics-server addon must be enabled in the Minikube cluster in order to retrieve this information.

5.5 Testing Scenarios

To assess the effectiveness of the proposed carbon aware scheduling approach, we will be evaluating the total CO_2 savings that are achieved through the simulation of workloads and monitoring of energy consumption in the cluster. To provide a baseline for comparison, we will also be evaluating the performance of the standard implementation of the K8s scheduler. The carbon emissions will be calculated using the cluster's energy consumption and the gCO_2eq/kWh of the grid. The cluster's energy consumption will be estimated using a linear power model that considers the CPU utilization. This allows us to accurately compare the performance of the carbon aware scheduler with the standard K8s scheduler and evaluate the potential for reducing carbon emissions in datacenters.

For the first testing scenario 5.6, we used the LCG dataset to generate the workload. The utilization of the LCG cluster is low and evenly distributed. The duration of the test was 24 hours, and the test was initiated at 5 pm in order to maximize shifting potential and demonstrate the effectiveness of the proposed scheduling approach. This testing scenario allows us to evaluate the performance of the carbon aware scheduler in a low-utilization environment and assess its potential for reducing carbon emissions in datacenters.

For the second testing scenario, we used the SHARCNet dataset to generate the workload. Unlike the LCG dataset, the SHARCNet dataset has a higher utilization and is less evenly distributed. The duration of the test was also 24 hours, and the test was initiated at 5 pm in order to maximize shifting potential. This testing scenario allows us to evaluate the performance of the proposed carbon aware scheduler in a medium-utilization environment and assess its potential for reducing carbon emissions in datacenters.

For both testing scenarios, the CO_2 prediction utilized CO_2 signals from Germany in 2018 as training data.

For the third testing scenario, we used the LCG dataset to evaluate the performance of the proposed Spatial Workload Shifter. In this test, we simulated two nodes in a Minikube cluster, each representing a different location with similar but alternting grid carbon emissions. The emission

data from France and Brazil were chosen, due to their similar average carbon emissions and alternating optimal CO_2 windows. The CO_2 emission curves of the two nodes can be viewed in figure 5.6b.

The objective of the fourth testing scenario was to evaluate the performance of the Spatial Workload Shifter in a cluster with nodes located in distinct geographical regions, each with varying grid carbon emissions. The emission data used for this evaluation were sourced from Germany and France. Given the significant difference in CO_2 emissions between the two locations, it is anticipated that the Workload Shifter will demonstrate high efficiency in this scenario. The workload used in this test was generated using the SHARCNet dataset, and the CO_2 emission curves of the two nodes can be observed in Figure 5.7e.

In the fifth and sixth testing scenarios, the two scheduling methods were combined to maximize the potential savings.

The objective of the fifth scenario is to assess the effectiveness of the proposed scheduling method in a scenario of moderate utilization, where multiple locations with similar, but varying gCO_2eq/kWh values are present. The workload is generated using the SHARCNet dataset, and the nodes are configured with emission data from France and Brazil, as illustrated in Figure 5.6c.

The sixth scenario aims to evaluate the effectiveness of the proposed scheduling method in a scenario of moderate utilization, where multiple locations with distinct gCO_2eq/kWh values are present. The workload is generated using the SHARCNet dataset, and the nodes are configured with emission data from France and Germany, as illustrated in Figure 5.7e.

5.6 Evaluation

In this section, we present the results of the evaluation of the various testing scenarios. The scenarios are labeled based on their respective method, location, and workload.

Testing Scenario 1 - Temporal, Germany, Low utilization

As previously mentioned in Section 5.5, the LCG dataset represents a low utilization workload scenario. The CPU reservation curve for this scenario is shown in Figure 5.2a, with the cyan curve representing the standard Kubernetes implementation and the magenta curve representing the Temporal Workload Scheduler.

It can be observed that the Temporal Workload Scheduler scheduled the workload one hour too early due to an inaccurate prediction of CO_2 emissions. However, as shown in Figure 5.2c, which compares the CO_2 prediction (purple curve) to the actual CO_2 curve (blue curve), there is not a significant difference between the optimal hour and the second most optimal hour. As a result, the performance drawback is minimal.

The energy consumption for the scenario is shown in Figure 5.2b. It was calculated using the linear power model, which is depicted in Figure 5.1 (p.60 [Pio22]). The CO_2 emissions per hour, depicted in Figure 5.2d, were calculated by combining the energy consumption data with CO_2 emission data, and are expressed in $gCO2/h$.

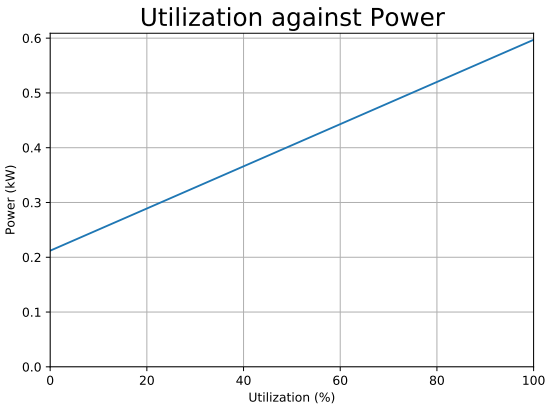
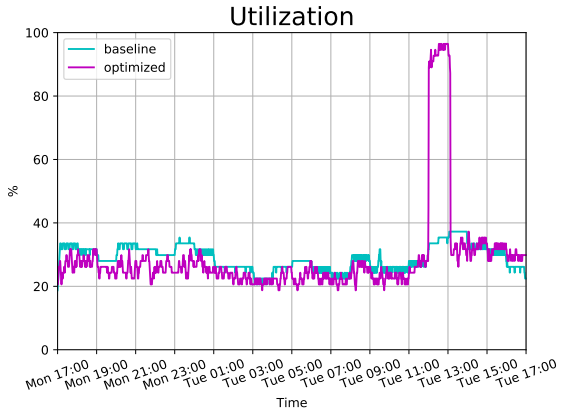


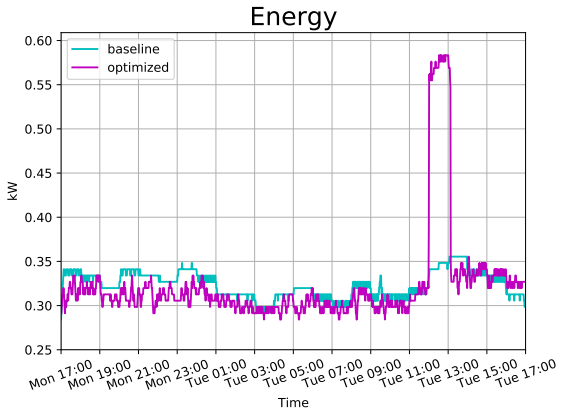
Figure 5.1: Power Model

In Figure 5.2e, the total CO_2 emissions for the standard implementation and the Temporal Workload Scheduler in the first scenario are depicted and expressed as total gCO_2 at hour h. The green curve representing the Temporal Workload Scheduler grows at a slower rate than the red line, indicating a reduction in carbon emissions. However, at approximately 12:00, there is a increase in the total CO_2 emissions due to the Workload Scheduler increasing its utilization in the CO_2 optimal hour. After this point, the two curves continue with a similar slope, indicating that both implementations make similar scheduling decisions. Nonetheless, a gap between the two curves can still be observed, representing the total carbon emission savings achieved by the Temporal Workload Scheduler.

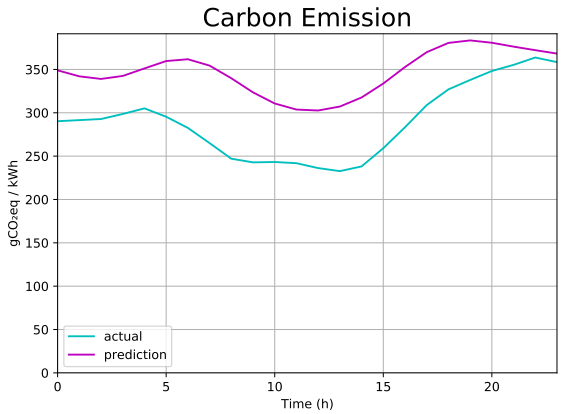
To more clearly illustrate the gap in total CO_2 emissions, the difference between the two curves in Figure 5.2e is depicted in Figure 5.2f. As shown in this figure, the Temporal Workload Shaper achieved a carbon saving of approximately 20 g. Given that the total emissions of the workload scenario are approximately 2245 g, this represents a carbon emission reduction of **0.85%**.



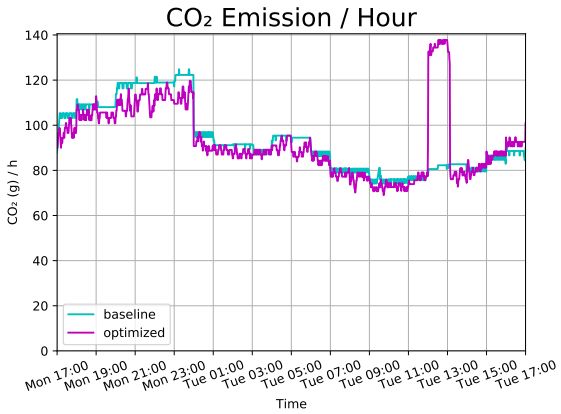
(a) Cluster CPU Reservation



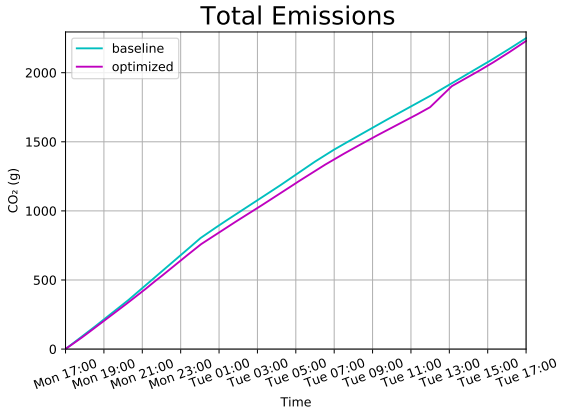
(b) Cluster Power Consumption



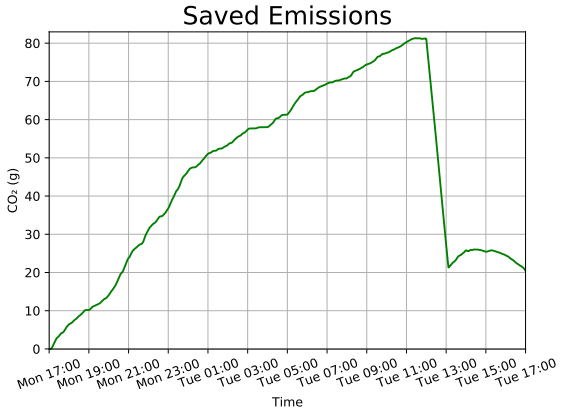
(c) Cluster CO_2 Prediction vs. Actual CO_2



(d) CO_2 Emission Rate



(e) Total CO_2 Emissions



(f) Saved CO_2 Emissions

Figure 5.2: First Scenario

As previously mentioned in Section 5.3 the SLA compliance was measured by counting the amount of pending critical jobs each minute. If no resources are available for critical workloads, the number of pending critical jobs will increase, indicating a violation of the SLA The Figure 5.3 presents a kernel density chart of the number of pending critical jobs for both baseline and optimized methods

and demonstrates that both implementations exhibit equal compliance with the SLA. The minor peak at 1 does not necessarily imply a violation but could be a result of a job being scheduled just before the logger polled the queue status. However, a peak at 2 or higher would indicate congestion, as multiple jobs are not scheduled simultaneously in our workload scenario.

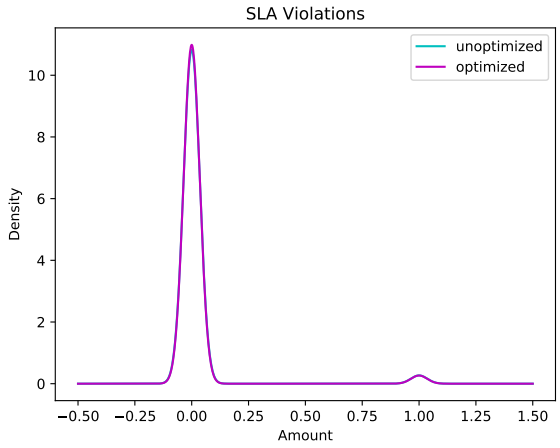
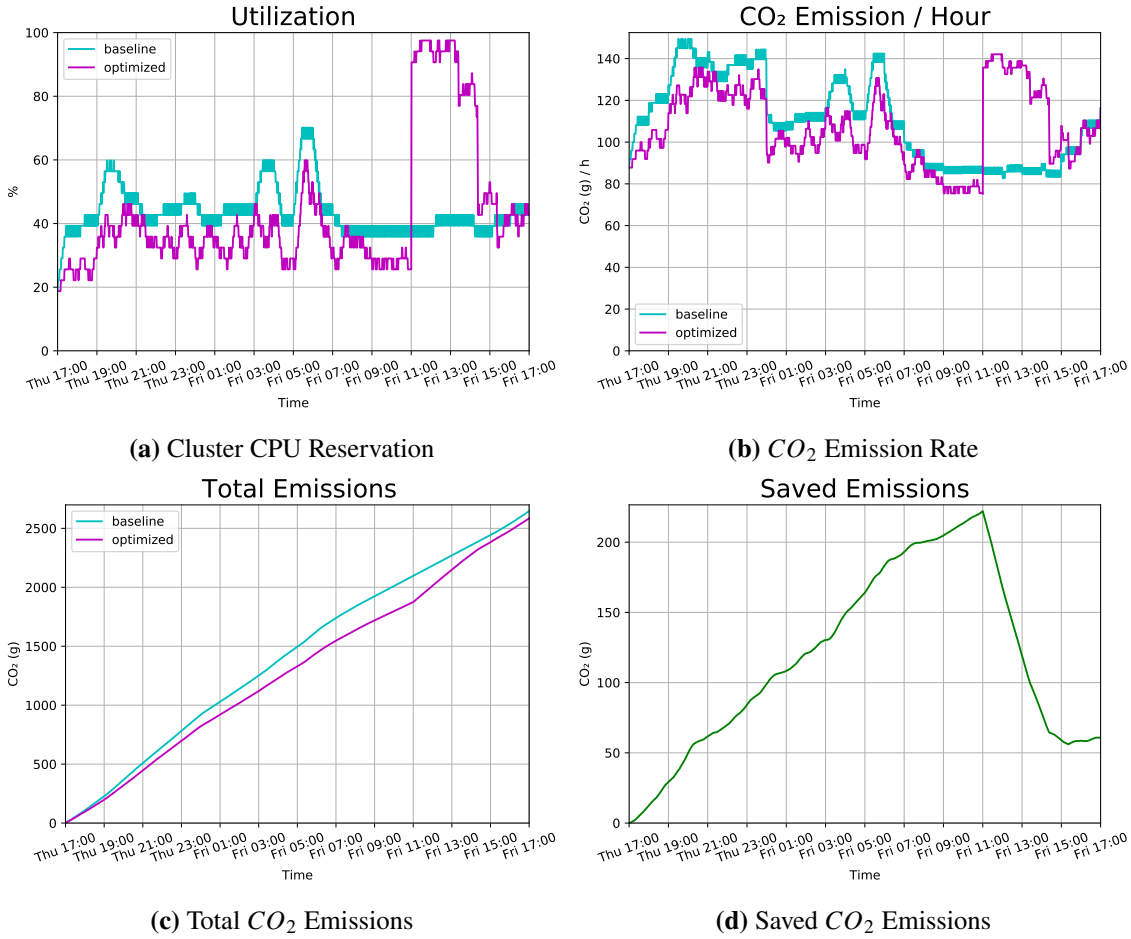


Figure 5.3: SLA Violations

Testing Scenario 2 - Temporal, Germany, Medium utilization

For the testing scenarios with medium utilization, we used the SHARCNet dataset. The carbon emissions data for this scenario were taken from the first testing scenario, as shown in Figure 5.2c. The CPU reservation curve for this scenario is depicted in Figure 5.4a. It can be observed that the optimal window determined by the scheduler is significantly larger compared to the first testing scenario, which used an overall low utilization workload. Figure 5.4b shows the CO2 emissions per hour, which were calculated using the power consumption data and the grid carbon emissions data.

The total CO_2 emissions for the second testing scenario, as shown in Figure 5.4c, exhibit a larger gap compared to the first testing scenario. This can be attributed to a higher utilization level, which results in a lower proportion of static energy consumption and a higher relative shift of energy consumption to more optimal hours. The second testing scenario achieved total savings of 60.82 g CO2, or a saving of **2.35%** of the total emissions of 2645 g, as seen in Figure 5.4d.



(a) Cluster CPU Reservation

(b) CO_2 Emission Rate

(c) Total CO_2 Emissions

(d) Saved CO_2 Emissions

Figure 5.4: Second Scenario

A visual representation of the density of SLA violations can be observed in Figure 5.5. It should be noted that, once again, the optimized method did not result in a impairment of SLA compliance.

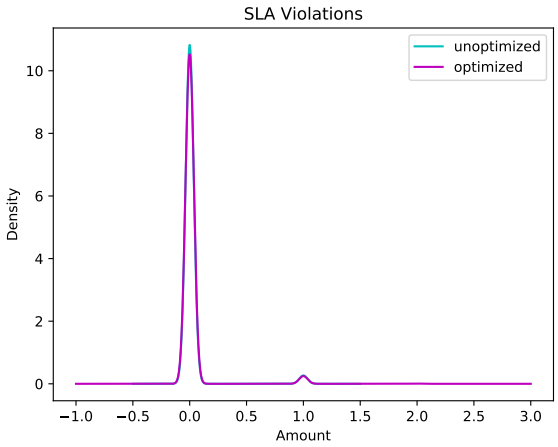


Figure 5.5: SLA Violations

Testing Scenario 3 - Spatial, France Brazil, Low utilization

The objective of this testing scenario was to evaluate the effectiveness of the Spatial Workload Shifter method using emission data from France and Brazil. The LCG workload was utilized for a low utilization scenario and the distributed cluster consisted of two nodes, one master node hosting the control plane and one worker node. The goal was to determine the potential carbon savings in a distributed cluster with two similar and alternating CO_2 emission curves. The simulation of the carbon emission API was conducted using historical gCO_2eq/kWh data that was available.

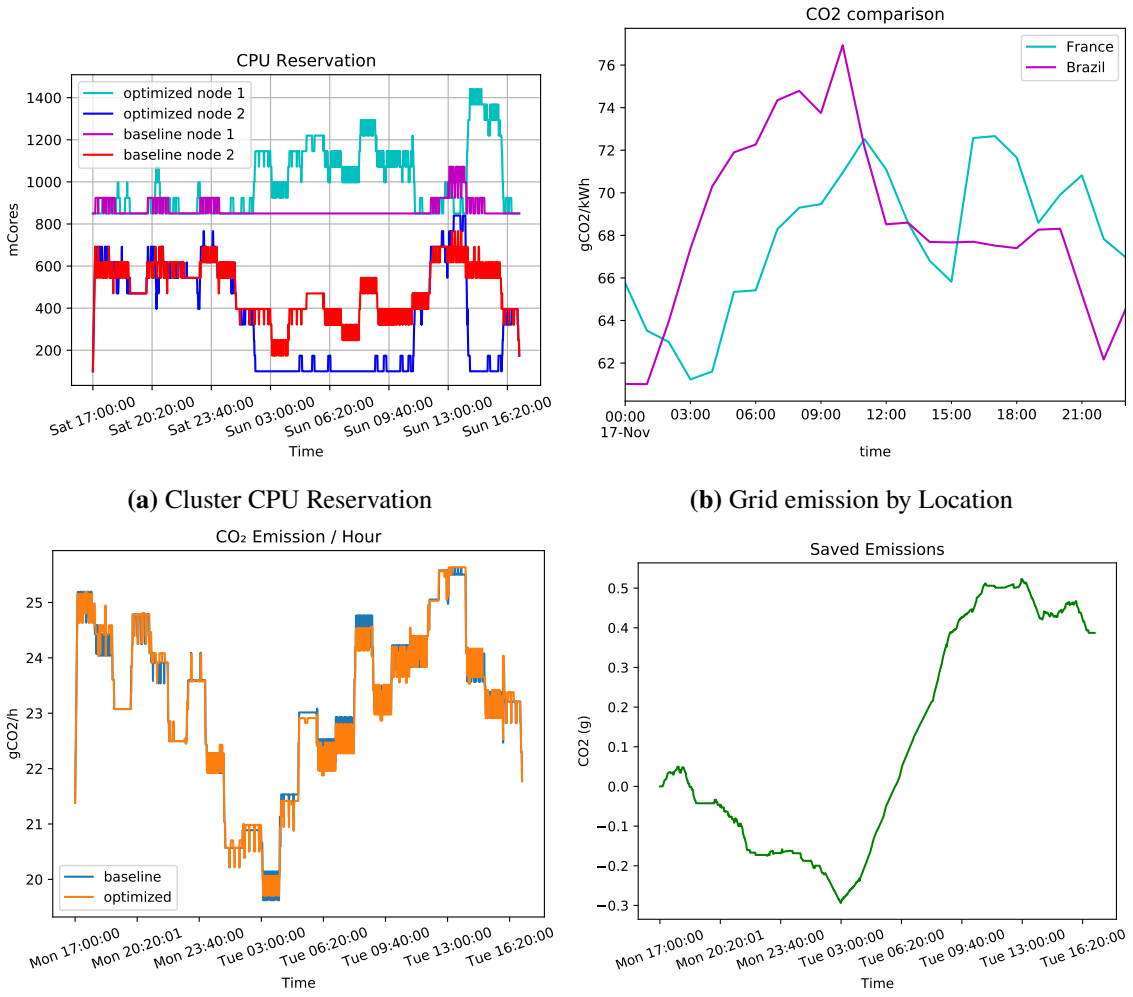
As presented in Figure 5.6a, the CPU reservation of the different nodes is illustrated. It is apparent that the baseline implementation emphasizes on maintaining equal utilization of the nodes. In contrast, the optimized method prioritizes deploying the workload on the node with the lowest carbon emissions, regardless of the nodes' utilization.

It was observed that both clusters have identical energy consumption at all times. This is a result of the fact that the workload is executed immediately and not delayed in this method, unlike the temporal approach.

The gCO_2eq/kWh values for this test scenario are presented in Figure 5.6b. The similarity of the two emission curves indicates that the potential savings are minimal.

As depicted in Figure 5.6c, the CO2 emissions per hour can be observed. The Spatial Workload Shifter method takes advantage of the differences in gCO_2eq/kWh between the nodes to minimize emissions. However, during the first quarter of the testing scenario, the optimized scheduler displayed higher emissions than the baseline implementation. This is due to the fact that the Workload Shifter only accounts for the current value of gCO_2eq/kWh. The overall emissions can exceed those predicted if the emissions in the next hour are significantly increasing and the job is still being executed. This problem can be addressed by utilizing runtime estimation and a gCO_2eq/kWh prediction. However, to prevent false scheduling, the predictions must be highly accurate.

As demonstrated in Figure 5.6d, the Spatial Workload Shifter method achieved only minimal savings. This outcome is consistent with the expectation, due to the similarity observed in Figure 5.6b. Nonetheless, the Workload Shifter was still able to take advantage of the existing fluctuations and reduce emissions in this scenario.



(a) Cluster CPU Reservation

(b) Grid emission by Location

(c) CO_2 Emission Rate

(d) Saved CO_2 Emissions

Figure 5.6: Third Scenario

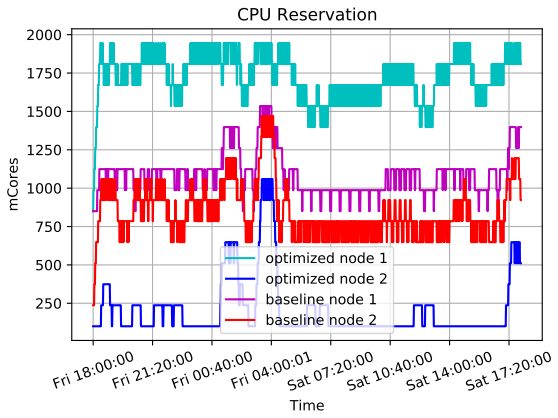
Testing Scenario 4 - Spatial, France Germany, Medium utilization

The objective of this testing scenario was to evaluate the efficacy of the Spatial Workload Shifter method using emission data from France and Germany. The SHARCNet workload scenario was utilized to simulate a medium-utilization environment. The cluster consisted of two nodes: one master node that hosts the control plane and also functioned as a worker node, and one worker-only node. The two nodes were simulated to be located in different locations with distinct grid emissions. The worker node used the emission data from Germany, while the master node used the emission data from France. As shown in Figure 5.7e, the two different grid emission curves are illustrated. It is apparent that the difference between the two emission curves is substantial, indicating a high potential for reducing CO_2 emissions.

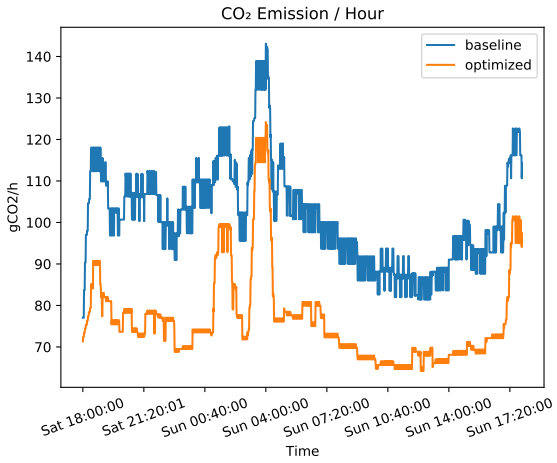
As illustrated in Figure 5.7a, the CPU reservation in milli cores per node can be observed. The optimized implementation maximizes the utilization of node 1, which is simulated to be located in France. In contrast, the baseline method aims to evenly distribute the utilization between both nodes.

The cluster emissions can be observed in Figure 5.7b. It is apparent that the optimized cluster consistently outperforms the baseline method. The total CO_2 emissions, as displayed in Figure 5.7c, reveal a significant difference between the two implementations.

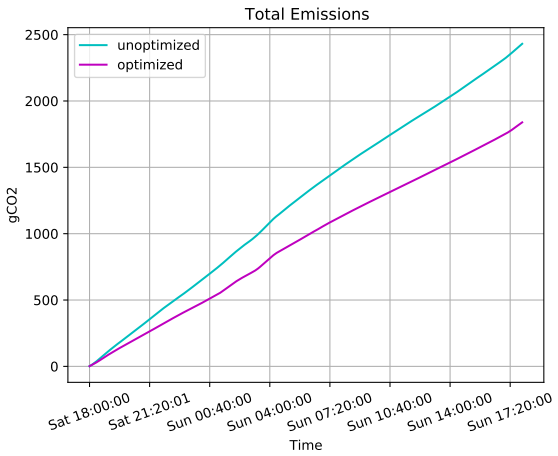
In Figure 5.7d, the total saved emissions are presented. It can also be observed that the savings are linear, this is because the Workload Shifter does not delay workload, but rather shifts it to less carbon-intensive locations. The Spatial Workload Shifter method achieved savings of $592g$, which corresponds to a percentage saving of **32.19%**.



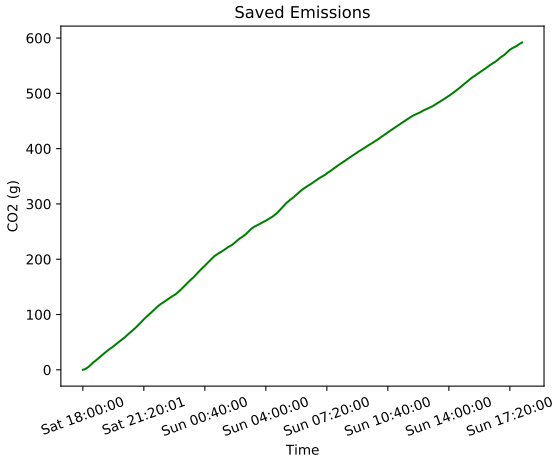
(a) Cluster CPU Reservation



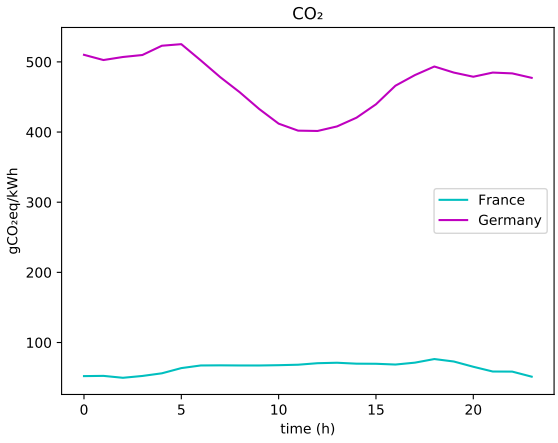
(b) CO_2 Emission Rate



(c) Total CO_2 Emissions



(d) Saved CO_2 Emissions



(e) Grid emission by location

Figure 5.7: Forth Scenario

Testing Scenario 5 - Temporal + Spatial, France Brazil, Low utilization

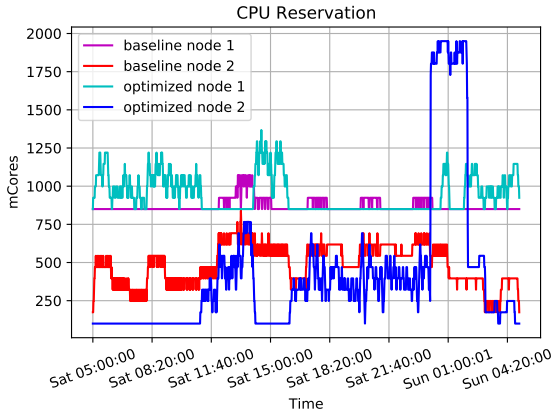
The goal of this testing scenario was to evaluate the effectiveness of the combined scheduling method, utilizing both the Spatial Workload Shifter and the Temporal Workload Scheduler, under low utilization conditions and in geographical locations with similar and alternating gCO_2eq/kWh values, which can be observed in Figure 5.6b. The test environment is similar to the one in Scenario 3, except the combined method was used in this scenario. It was expected that the combined scheduling method would perform at least as well as the Spatial Workload Shifter in Scenario 3.

As depicted in Figure 5.8a, the CPU reservation of both nodes in the cluster can be observed. It is clear that the emission of node 1 is lower than the emission of node 2 during certain time periods, specifically between 00:00 and 11:00 and again from 14:00 to 16:00. The majority of the workload is scheduled on the second node during the afternoon and evening. This behavior aligns with the observations made in Scenario 3. However, due to the implementation of the Temporal Workload Scheduler, non-critical workload has been postponed to the hours of 23:00 and 00:00 on the second node, which were predicted to be the most optimal in terms of CO_2 emissions.

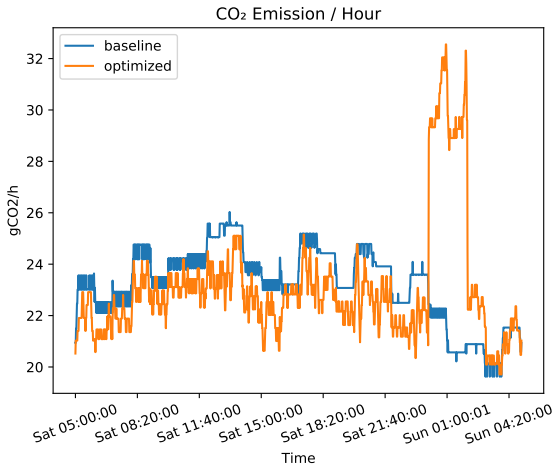
In Figure 5.8b, the hourly CO_2 emissions can be observed. The optimized method exhibits improved performance compared to the baseline method. Additionally, a spike in emissions can be noted at 23:00 and 00:00, which is a result of the Temporal Scheduler.

The total emissions savings achieved through the optimization method can be viewed in Figure 5.8d and indicate a minor reduction compared to Scenario 1 and 2, with a total saving of 2.56g of CO_2, which constitutes a **0.46%** reduction in this scenario. It should be noted, however, that the combined method produced higher savings compared to the approach in Scenario 3.

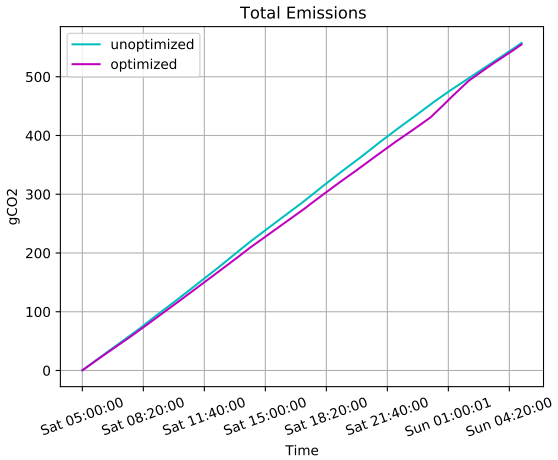
A visual representation of the density of SLA violations can be observed in Figure 5.8e. Both the optimized and baseline methods display comparable behavior, indicating that the optimized method did not result in a worsening of SLA compliance.



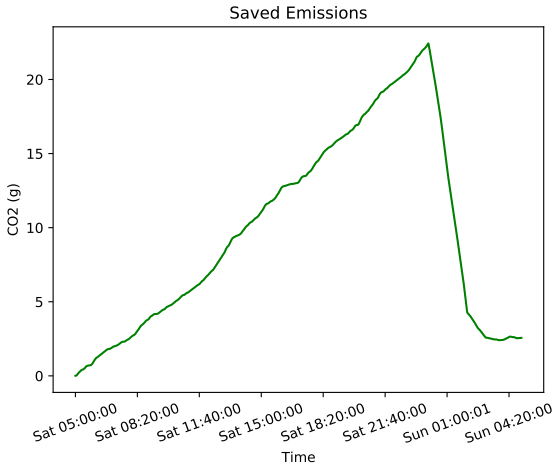
(a) Cluster CPU Reservation



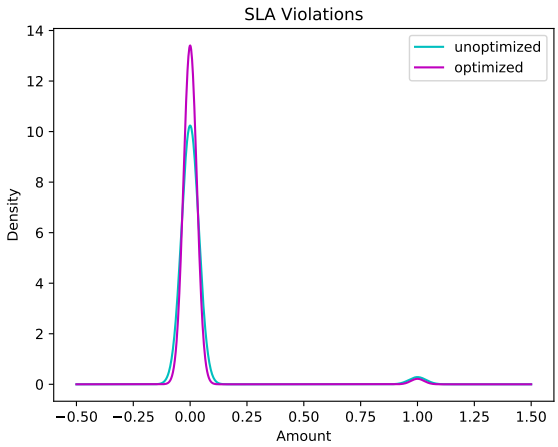
(b) CO_2 Emission Rate



(c) Total CO_2 Emissions



(d) Saved CO_2 Emissions



(e) SLA Violations

Figure 5.8: Fifth Scenario

Testing Scenario 6 - Temporal + Spatial, France Germany, Medium utilization

The objective of this testing scenario was to assess the effectiveness of the combined scheduling method, under medium utilization conditions and in two geographical locations with different gCO_2eq/kWh values. It was expected that the combined scheduling method would perform at least as well as the Spatial Workload Shifter in Scenario 4. However, the results indicated that the combined scheduling method performed suboptimally, primarily due to inaccurate predictions. Additionally, the fluctuation of gCO_2eq/kWh at the node simulated to be located in France was not significant enough to be effectively utilized.

The cluster CPU reservation data illustrates several spikes for node 2 between 22 : 00 and 5 : 00. This outcome was expected to some extent, as similar spikes were observed in the fourth scenario (Figure 5.7a), which is comparable except that the Temporal Workload Scheduler was not used. However, the frequency and magnitude of these spikes were found to be significantly higher than in the fourth scenario. This was determined to be a result of the inaccurate workload prediction of the SHARCNet workload. As depicted in Figure 4.18, it can be observed that the predicted workload between the hours 0 and 5 was much lower than the actual workload for this time window. This inaccurate prediction led to the Temporal Workload Scheduler scheduling more non-critical workload to these hours on the optimal node than the resources available, resulting in the unavailability of resources for critical workload on the optimal node at this point in time, ultimately leading to the scheduling of critical workload on the less optimal node, thereby increasing the overall emissions.

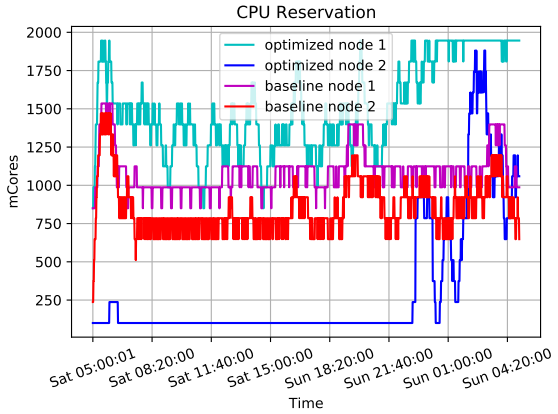
One solution to mitigate resource contention in a system is to implement preemptive scheduling, which allows for the interruption of non-critical workload in order to allocate resources to critical workload. However, it is important to note that this approach requires interruptible workloads, as otherwise progress may be lost and overall energy consumption may increase upon redeployment of the workload.

Another approach to reduce emissions in this scenario is to increase the resources available in the cluster. By increasing the number of millicores per node, more flexible workload scheduling can be achieved, which would be more resilient to inaccurate predictions. This would result in greater emissions savings.

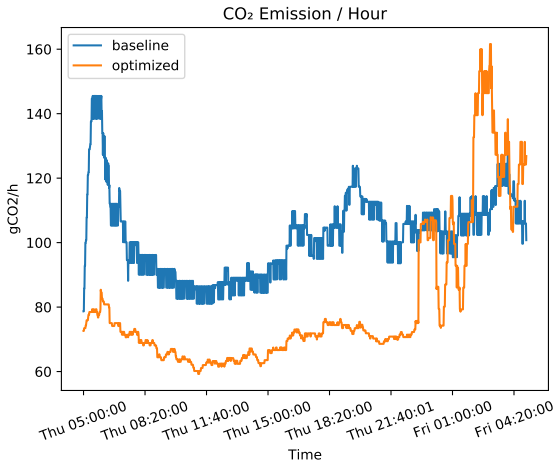
Additionally, as depicted in Figure 5.7e, it is evident that the variation in the emission curve of France is significantly less compared to that of Germany. As a result, the potential for carbon savings in this scenario is significantly lower in comparison to scenarios 1 and 2, where the Temporal Workload Scheduler was implemented on a single node simulated to be located in Germany.

The total saved emissions of this scenario were found to be around $515g$, which corresponds to a percentage saving of **26.78%**, as seen in Figure 5.9d.

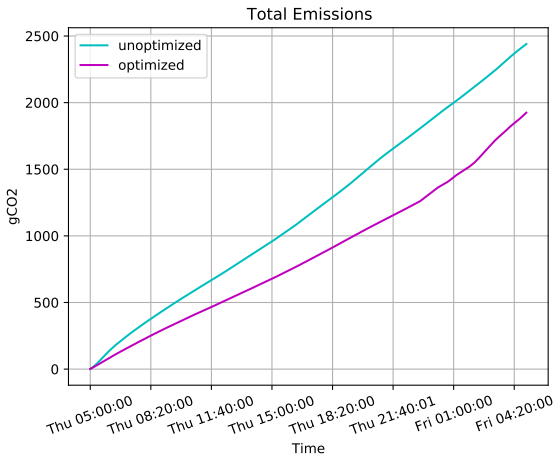
A visual representation of the density of SLA violations can be observed in Figure 5.9e. It should be noted that, once again, the optimized method did not result in a worsening of SLA compliance.



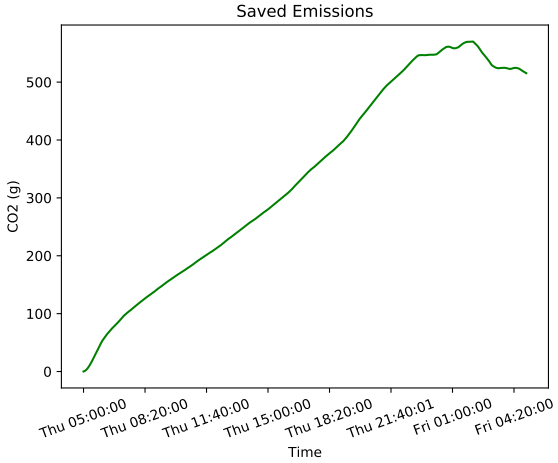
(a) Cluster CPU Reservation



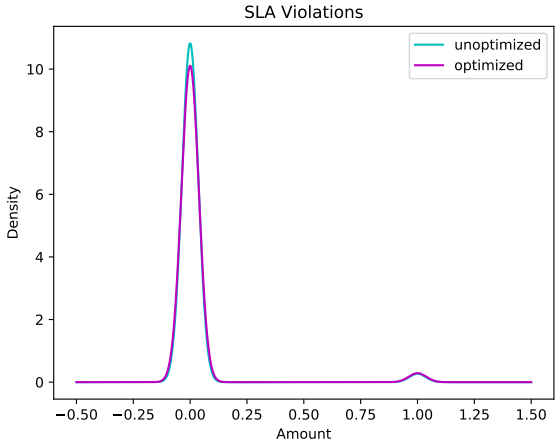
(b) CO_2 Emission Rate



(c) Total CO_2 Emissions



(d) Saved CO_2 Emissions



(e) SLA Violations

Figure 5.9: Sixth Scenario

5.7 Challenges and Limitations

It is important to acknowledge the limitations set in the testing environment due to the potential complexity of some scenarios. The testing environment only considers two types of workload, namely critical and non-critical workload. In real-world applications, workload may be bound to specific locations, thus requiring constraints in space in addition to time. Additionally, only the CPU reservation is considered when scheduling workloads, this is due to the unavailability of reliable memory usage data in the used datasets.

Furthermore, the inflexibility of long-running non-interruptable workloads may potentially impact the performance of the scheduler. To demonstrate the beneficial effects of the scheduler on long-running workloads, tests exceeding 24 hours would be required. To mitigate this, the tests were limited to a maximum execution time of 1 hour.

Another limitation of the 24-hour test execution is that all workloads must be deployed within this timeframe, including those scheduled in the last hour, leading to suboptimal behavior after the optimal hour for CO_2 emissions has passed. In a real-world scenario, workloads would only be limited to run in the next 24 hours from the time of creation. Therefore, workloads which are scheduled in the last few hours could be postponed to the next day's optimal hour.

6 Conclusion and Outlook

In conclusion, this thesis presents a novel approach to reducing carbon emissions in cloud computing by using statistical models and scheduling strategies in a K8s cluster. The use of statistical models such as the Prophet model showed promising results in predicting CO_2 emissions, runtime and resource usage per job. However, it was found that these models do not work as well in predicting the total amount of incoming workload per hour. The trend analysis presented in Figure 4.7 demonstrates that a shift of workload towards the weekend can result in significant increase in CO_2 savings, which aligns with the conclusions drawn by Wiesner et Al. in their publication (p.6 [WBS+21]).

Both the temporal and spatial methods achieved noticeable carbon savings. We found, that the performance of the Temporal Workload Scheduler depends on the accuracy of the CO_2 and workload predictions and is highly susceptible to incorrect predictions. On the other hand, the Spatial Workload Shifter performs best in environments with high differences in CO_2 emissions between nodes and does not rely on predictions, making it suitable for highly unpredictable workloads. Furthermore, the trade-off between efficiency and SLA should also be considered when deciding on which method to use. The combined scheduling method performs best in multi-location clusters with distinct grid emissions, which also show a high daily variance in gCO_2eq/kWh values.

In light of these findings, it is evident that there is potential for significant carbon emission savings in cloud computing through the use of intelligent scheduling strategies.

Outlook

Going forward, there are several areas that can be explored to further improve the performance of the scheduler. The use of preemptive scheduling can be considered to improve performance of the temporal scheduler and overall SLA compliance. Preemptive scheduling could potentially decrease the impact of inaccurate workload predictions, by preempting non-critical workload for higher SLA compliance and efficiency. The exploitation of interruptible workload can also be investigated to optimize the use of resources, which would enable the scheduler to make more optimal decisions, especially for long-running workloads. In addition, finding a better way to predict the total number of jobs per hour is an important area of improvement, this could increase the workload prediction accuracy, which would increase the efficiency of the Temporal Workload Scheduler. Finally, deploying the scheduler in a real-world scenario will provide valuable insights into its performance and effectiveness in reducing carbon emissions.

Bibliography

[05a] In: 2005. URL: http://gwa.ewi.tudelft.nl/datasets/gwa-t-11-lcg (cit. on p. 34).

[05b] In: 2005. URL: http://gwa.ewi.tudelft.nl/datasets/gwa-t-10-sharcnet (cit. on p. 35).

[Age] U. S. E. P. Agency. In: URL: https://www.epa.gov/climate-indicators/greenhouse-gases (cit. on p. 17).

[CNB21] CNBC. In: 2021. URL: https://www.cnbc.com/2021/11/04/gap-between-renewable-energy-and-power-demand-oil-gas-coal.html (cit. on p. 17).

[Doc] Docker. In: URL: https://www.docker.com/ (cit. on p. 22).

[Ene] EnergyX. In: URL: https://energyx.com/resources/what-is-intermittency-in-renewable-energy/ (cit. on p. 17).

[Gui19] E. T. Guinevere Saenger grodrigues3. In: 2019. URL: https://github.com/kubernetes/design-proposals-archive/blob/main/scheduling/scheduler_extender.md (cit. on p. 21).

[IRE17] IRENA. In: 2017. URL: https://www.irena.org/publications/2017/oct/electricity-storage-and-renewables-costs-and-markets (cit. on p. 17).

[ISE21] F. ISE. In: 2021. URL: https://strom-report.de/strom (cit. on p. 17).

[JS19] A. James, D. Schien. "A Low Carbon Kubernetes Scheduler". In: July 2019 (cit. on pp. 17, 23).

[Kub22a] Kubernetes. In: 2022. URL: https://kubernetes.io/ (cit. on p. 20).

[Kub22b] Kubernetes. In: 2022. URL: https://kubernetes.io/docs/concepts/scheduling-eviction/ (cit. on p. 20).

[Kub22c] Kubernetes. In: 2022. URL: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/ (cit. on pp. 20, 21).

[Map] E. Maps. In: URL: https://www.electricitymaps.com/ (cit. on pp. 27, 28, 44).

[Met] Meta. "Prophet | Forecasting at scale". In: URL: https://facebook.github.io/prophet/ (cit. on pp. 30, 32, 36).

[MTC+22] M. S. Misaghian, G. Tardioli, A. G. Cabrera, I. Salerno, D. Flynn, R. Kerrigan. "Assessment of Carbon-Aware Flexibility Measures from Data Centres using Machine Learning". In: *IEEE Transactions on Industry Applications* (2022), pp. 1–12. DOI: 10.1109/TIA.2022.3213637 (cit. on p. 25).

[Phi11] W. T. Philipp Wieder Joe M. Butler. "Service Level Agreements for Cloud Computing". In: Springer Science+Business Media, LLC., 2011, pp. 43–68. ISBN: 9781461416142 (cit. on pp. 17, 46).

[Pio22] T. Piontek. “CO2 Aware Job Scheduling for Data Centers”. In: 2022 (cit. on pp. 17, 23, 42, 48).

[RKS+22] A. Radovanovic, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, W. Cirne. “Carbon-Aware Computing for Datacenters”. In: 2022, pp. 1–1. DOI: 10.1109/TPWRS.2022.3173250 (cit. on pp. 17, 18, 24).

[sci] scikit-learn. “Skforecast Docs”. In: URL: https://joaquinamatrodrigo.github.io/skforecast/0.4.3/index.html (cit. on p. 33).

[Shm16] G. Shmueli. *Practical Time Series Forecasting: A Hands-On Guide [3rd Edition]*. Practical Analytics. Axelrod Schnall Publishers, 2016. ISBN: 9780997847932. URL: https://books.google.de/books?id=nheXDwAAQBAJ (cit. on p. 22).

[Tec] D. U. of Technology. In: URL: http://gwa.ewi.tudelft.nl/grid-workload-format (cit. on p. 34).

[Thu23] S. Thummala. In: (2023). URL: https://medium.com/@sreekanth.thummala/kubernetes-probes-readiness-liveness-startup-explained-2f0cc01918a1 (cit. on p. 19).

[Vio20] B. Violino. In: (2020). URL: https://www.cio.com/article/196255/containers-and-kubernetes-3-transformational-success-stories.html (cit. on p. 19).

[WBS+21] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, L. Thamsen. “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. Québec city, Canada: Association for Computing Machinery, 2021, pp. 260–272. ISBN: 9781450385343. DOI: 10.1145/3464298.3493399. URL: https://doi.org/10.1145/3464298.3493399 (cit. on pp. 24, 63).

[XGB] XGBoost. In: URL: https://xgboost.readthedocs.io/en/stable/ (cit. on p. 39).

All links were last followed on March 2, 2023.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature