Hasso Plattner Institut
Digital Engineering · Universität Potsdam

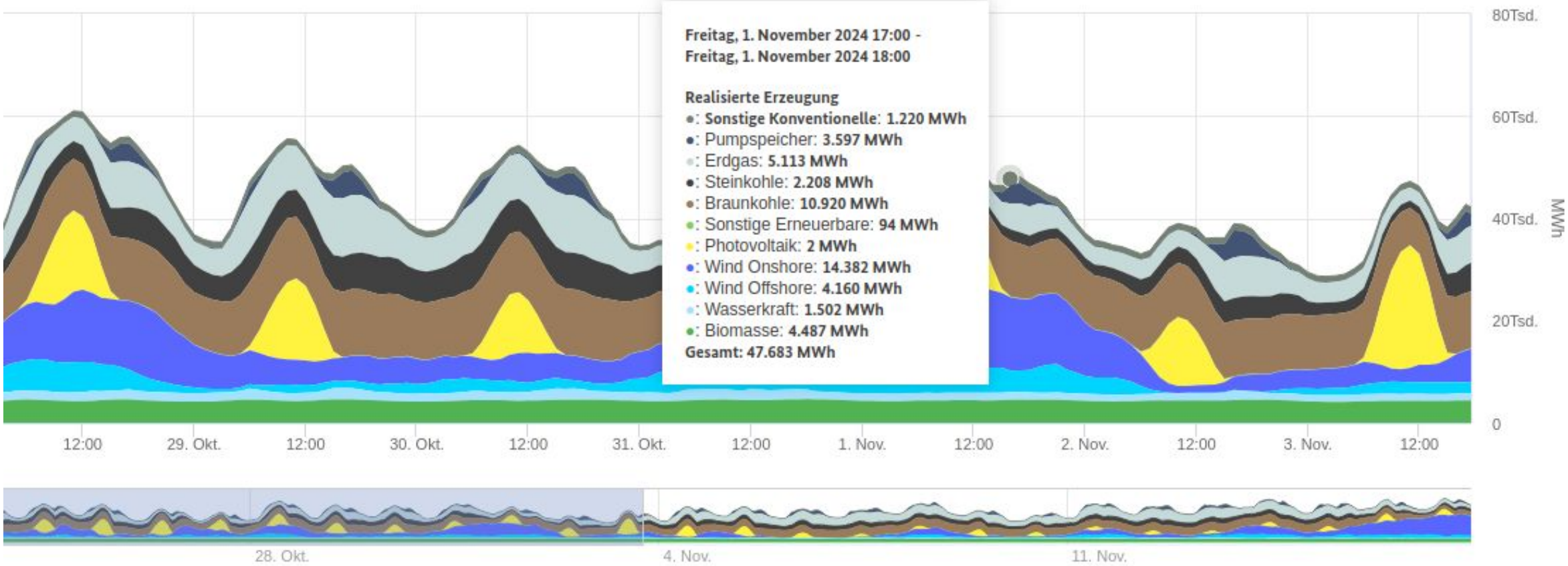
Masterarbeit Verteidigung

A Parameterizable Test Bed for Carbon Aware Job Scheduling

Design IT.
Create Knowledge.

www.hpi.de

17.12.2024



Stromproduktion aus versch. Quellen, smard.de/home/marktdaten

Ein immer größerer Anteil des Stroms in Deutschland wird durch Erneuerbare Energien gedeckt. Das zeigen die neuen Zahlen zum 1. Halbjahr 2024. Ein Überblick zum Ausbau von Solar- und Windenergie.

bundesregierung.de/breg-de/aktuelles/ausbau-erneuerbare-energien-2225808

Windräder auf See stehen häufiger still - vor allem vor SH

Stand: 25.04.2024 05:00 Uhr

Offshore-Windräder könnten deutlich mehr Strom nach SH liefern. Die Netzbetreiber schalten sie aber verstärkt ab, weil Leitungen überlastet sind. Das erhöht den Strompreis.

ndr.de/nachrichten/schleswig-holstein/Windraeder-auf-See-stehen-haeufiger-still-vor-allem-vor-SH,offshore924.html

tibber

Wusstest du schon: Wenn der Wind stark weht oder viel Sonne scheint, ist Energie besonders grün und günstig. Warum nicht dann deinen Strom verbrauchen? Tibber macht es dir einfach.

tibber.com

GLOBALE ERWÄRMUNG

Jeder einzelne der vergangenen zwölf Monate war weltweit der deutlich wärmste seiner Art

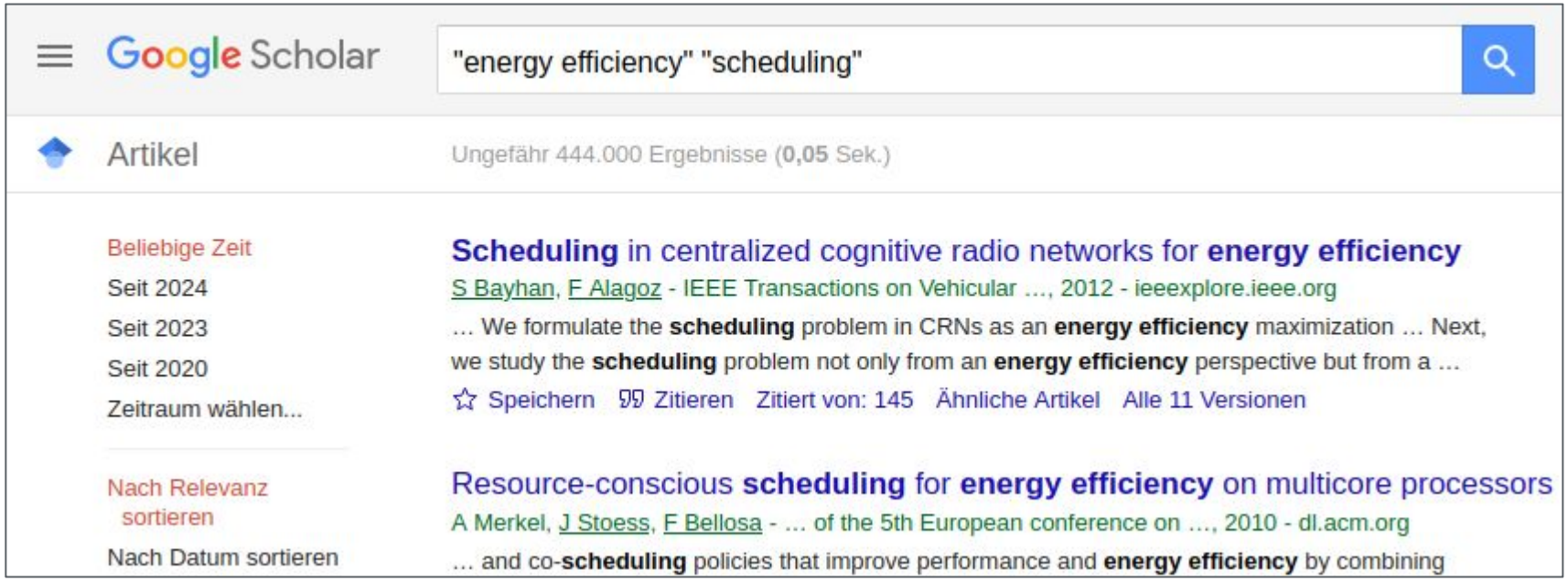
05. Juni 2024, 20:10 Uhr

mdr.de/wissen/umwelt-klima/waermste-zwoelf-monate-weltweit-waermster-meteorologischer-fruehling-deutschland-100.html

Strukturierte Literaturrecherche zum State of the Art

- Schlüsselwörter zu Nachhaltigkeit / Energieeffizienz und zu Rechenzentren

{ energy efficiency
energy consumption
carbon impact } × { datacenter, load balancing
scheduling, job shop
job management, compete cluster
hpc, cloud }



- jeweils 5 Abstracts lesen, Kategorisieren

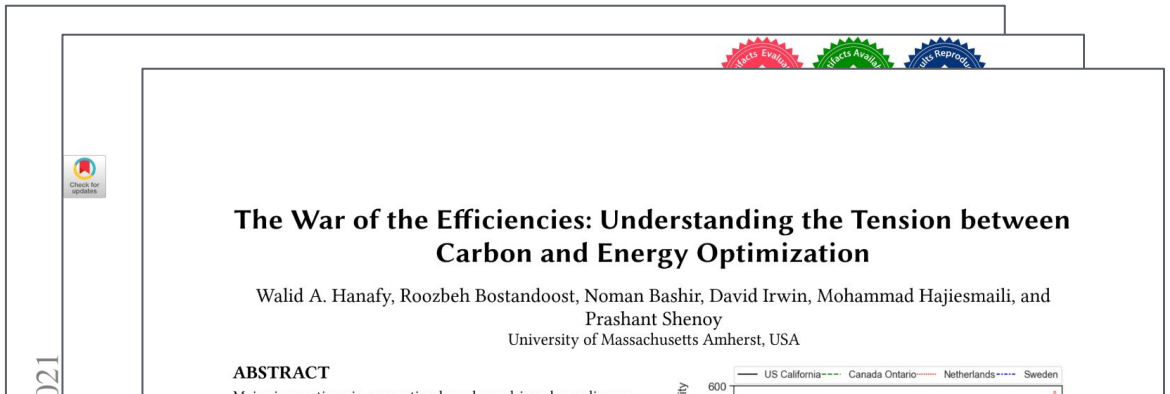
Strukturierte Literaturrecherche zum State of the Art

- 145 Paper erkundet
- davon 99 “rot / irrelevant”, 31 “orange / später”, und 15 “grün / komplett lesen, sehr relevant”

keyword 1	keyword 2	title
energy efficiency	datacenter	Beyond Energy-Efficiency: Evaluating Green Datacenter Applications for Energy-Agility
energy efficiency	datacenter	Energy Efficiency Dilemma: P2P-cloud vs. Datacenter
energy efficiency	datacenter	Energy Efficiency for Data Center and Cloud Computing: A Literature Review
energy efficiency	datacenter	Evaluation of the Heat and Energy Performance of a Datacenter Using a New Efficiency Index: Energy Usage Effectiveness Design – EUED.
energy efficiency	datacenter	Viability of datacenter cooling systems for energy efficiency in temperate or subtropical regions: Case study
energy efficiency	scheduling	Resource-conscious scheduling for energy efficiency on multicore processors
energy efficiency	scheduling	A review of energy-efficient scheduling in intelligent production systems
energy efficiency	scheduling	Integrating process optimization with energy-efficiency scheduling to save energy for paper mills
energy efficiency	scheduling	A genetic algorithm for energy-efficiency in job-shop scheduling
energy efficiency	scheduling	Optimal Scheduling of Demand Side Load Management of Smart Grid Considering Energy Efficiency
energy efficiency	compute cluster	GreenHDFS: Towards An Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster
energy efficiency	compute cluster	An Energy Efficiency Optimization and Control Model for Hadoop Clusters
energy efficiency	compute cluster	eeClust: Energy-Efficient Cluster Computing
energy efficiency	compute cluster	Energy efficient computing, clusters, grids and clouds: A taxonomy and survey
energy efficiency	compute cluster	An Empirical Study of Hadoop's Energy Efficiency on a HPC Cluster
energy efficiency	cloud	Managing overloaded hosts for energy-efficiency in cloud data centers
energy efficiency	cloud	Energy Efficiency for Cloud Computing Applications: A Survey on the Recent Trends and Future Scopes
energy efficiency	cloud	Adaptive Computational Solutions to Energy Efficiency in Cloud Computing Environment Using VM Consolidation
energy efficiency	cloud	A review of energy efficiency evaluation technologies in cloud data centers
energy efficiency	cloud	Energy efficiency in cloud computing data centers: a survey on software technologies
energy efficiency	load balancing	Energy efficiency of load balancing for data-parallel applications in heterogeneous systems
energy efficiency	load balancing	Energy Efficiency and Load Balancing in MANET: A Survey

State of the Art - Papers

Paper	Suspend & Resume Scheduling
Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud	✓
Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions	✓
The War of the Efficiencies: Understanding the Tension between Carbon and Energy Optimization	✓
On the Limitations of Carbon-Aware Temporal and Spatial Workload Shifting in the Cloud	✓
The Online Pause and Resume Problem: Optimal Algorithms and An Application to Carbon-Aware Load Shifting	✓



The War of the Efficiencies: Understanding the Tension between Carbon and Energy Optimization

Walid A. Hanafy, Roozbeh Bostandoost, Noman Bashir, David Irwin, Mohammad Hajiesmaili, and Prashant Shenoy

University of Massachusetts Amherst, USA

ABSTRACT

Probleme in Prior Work

- Ein Beispielhafter Machine Learning Workload

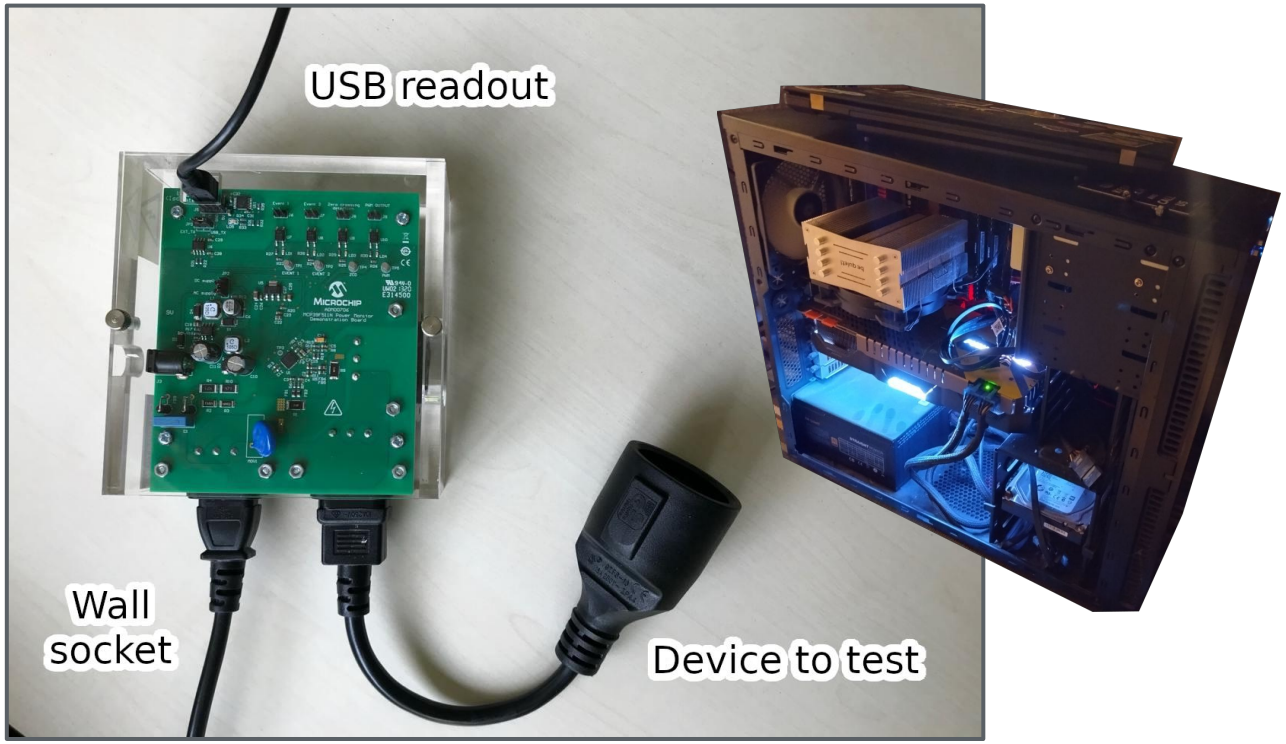
RoBERTa

Overview

The RoBERTa model was proposed in RoBERTa: A Robustly Optimized BERT Pretraining Approach by Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. It is based on Google's BERT model released in 2018.

It builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates.

Leistungsmessungen auf Machine Learning - Wie?

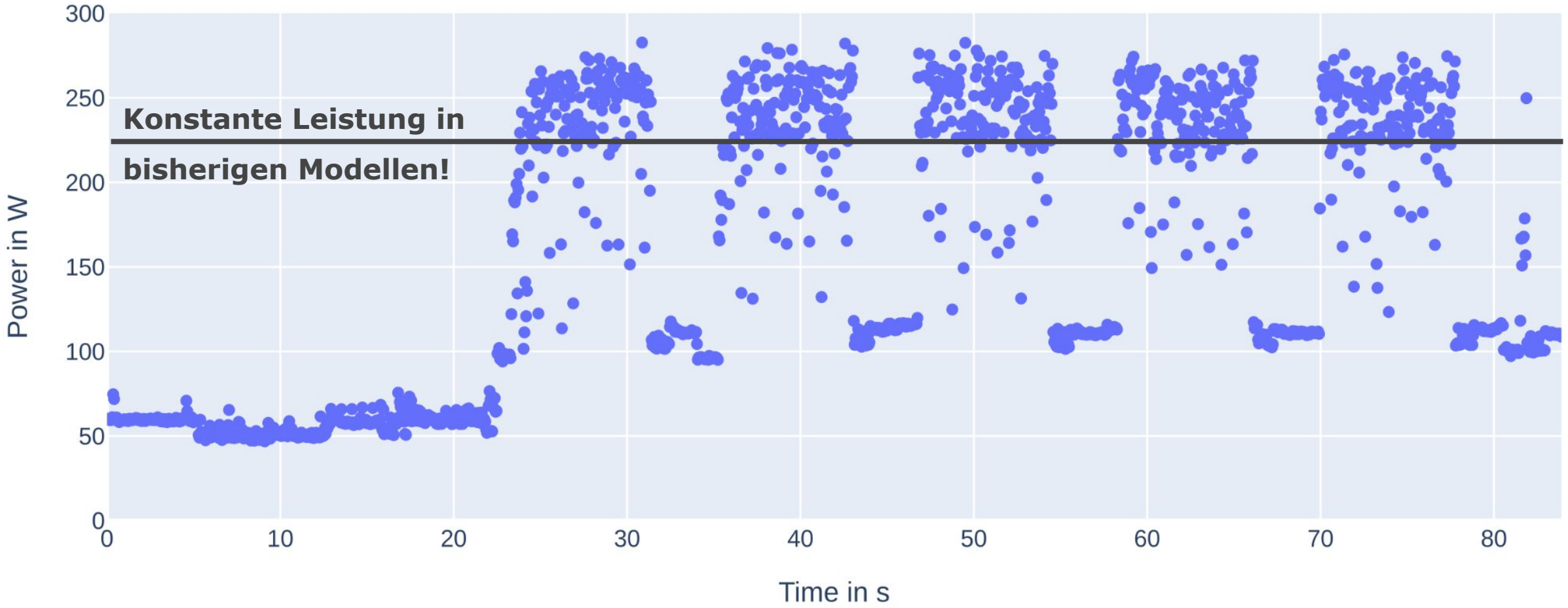


Microchip MCP39F511N Power Monitor

```
#!/bin/bash

# definiere Pfad zu Messprogramm pinpoint[1]
# setze CPU Frequenz
# setze Anzahl der Wiederholungen
# definiere Experimente
# Isoliere einzelne Runs voneinander
        # sleep
        # setze State zurück
echo “Done!”
```

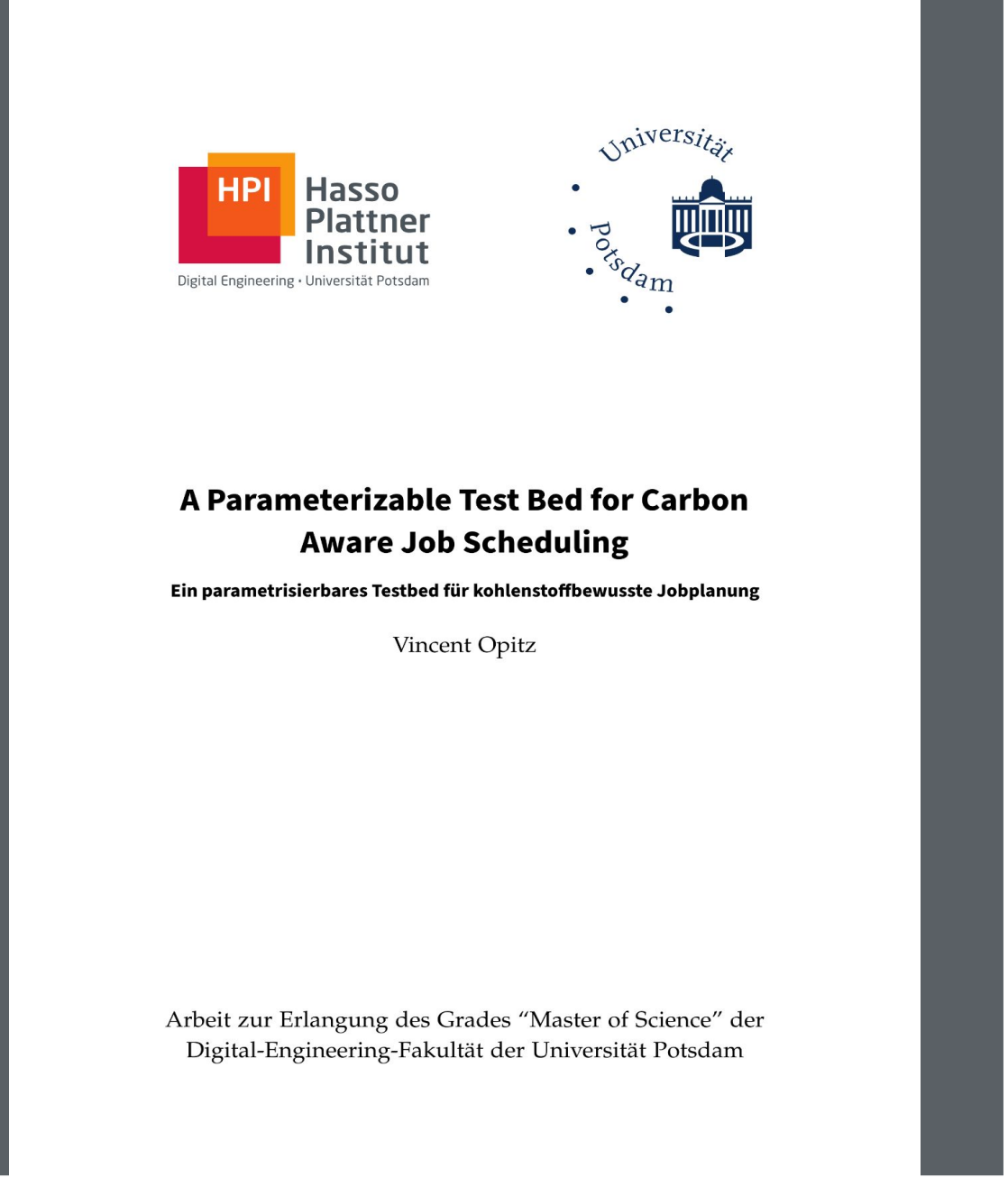
[1] github.com/osmhpi/pinpoint

Leistungsmessungen von RoBERTa



State of the Art - Papers

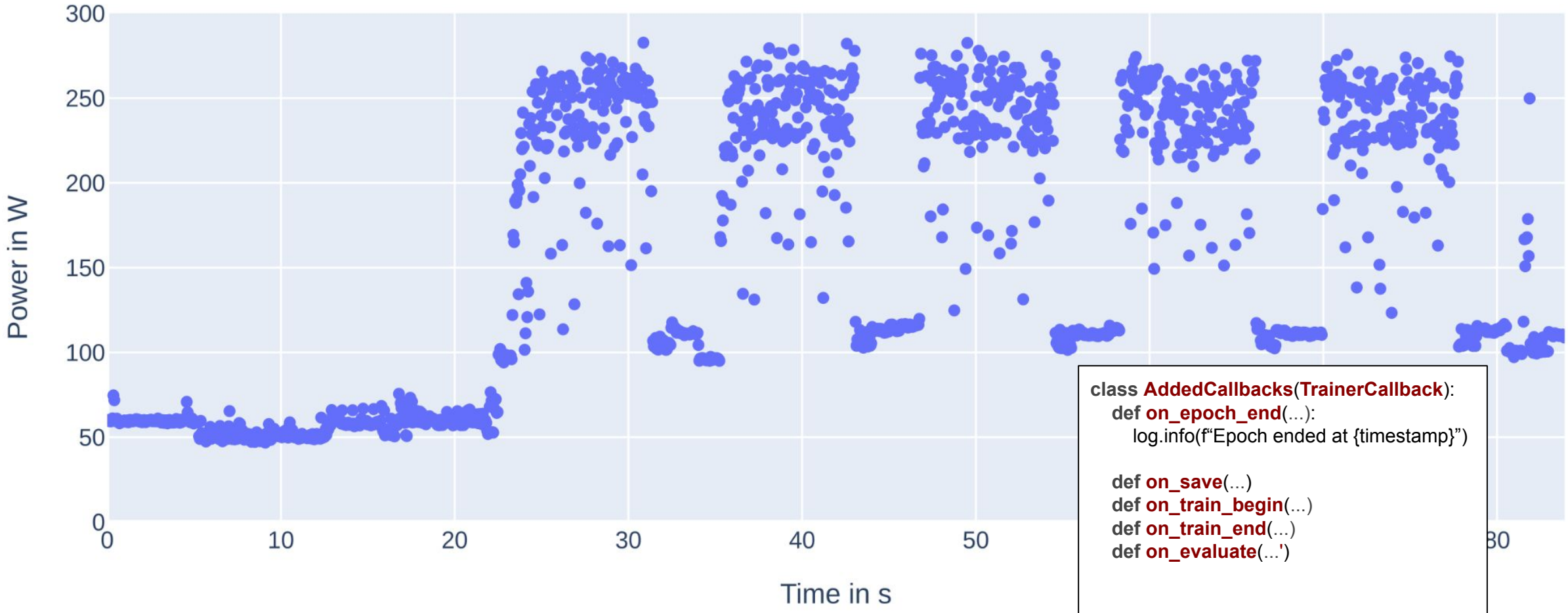
Paper	Suspend & Resume Scheduling	Covers Resume Overhead costs
Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud	✓	X
On the Limitations of Carbon-Aware Temporal and Spatial Workload Shifting in the Cloud	✓	X
The War of the Efficiencies: Understanding the Tension between Carbon and Energy Optimization	✓	~
Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions	✓	X
The Online Pause and Resume Problem: Optimal Algorithms and An Application to Carbon-Aware Load Shifting	✓	✓



Contributions / Ablauf

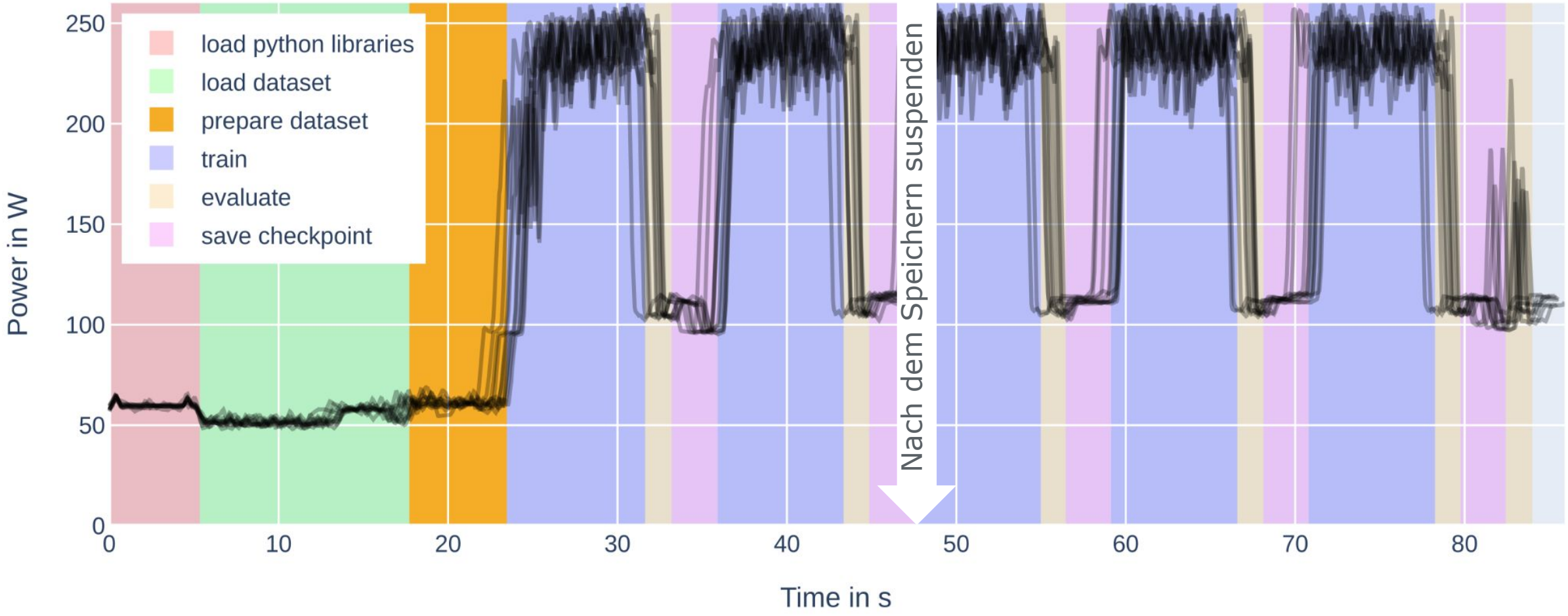
1. **Erstellen eines neuen Workload Modells**
   welches Resumption Overhead und Phasen beinhaltet
2. **Neue Scheduling Algorithmen**
   darauf basierend implementieren in einem Testbed
3. **Evaluation gegen das bestehende Modell**

Leistungsmessungen von RoBERTa




```
class AddedCallbacks(TrainerCallback):
    def on_epoch_end(...):
        log.info(f“Epoch ended at {timestamp}”)

    def on_save(...)
    def on_train_begin(...)
    def on_train_end(...)
    def on_evaluate(...')
```

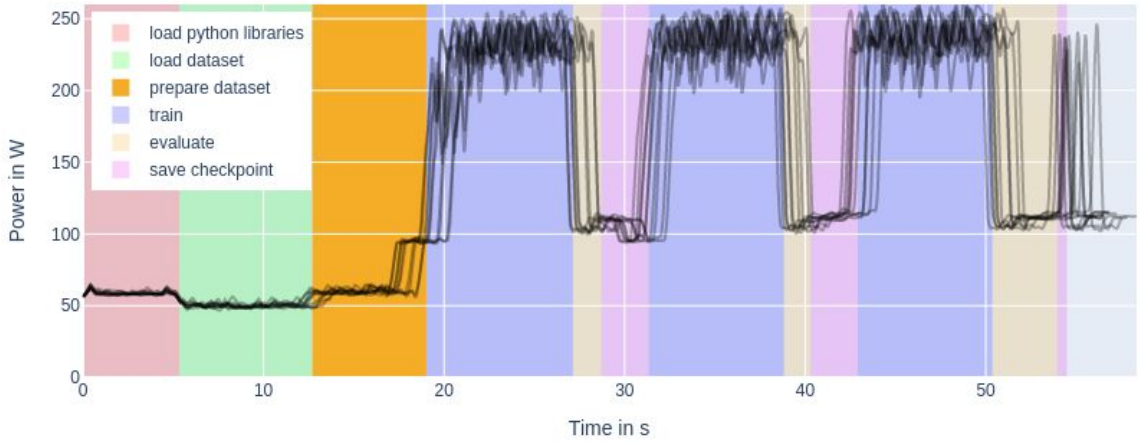
Leistungsmessungen von RoBERTa



Leistungsmessungen von RoBERTa

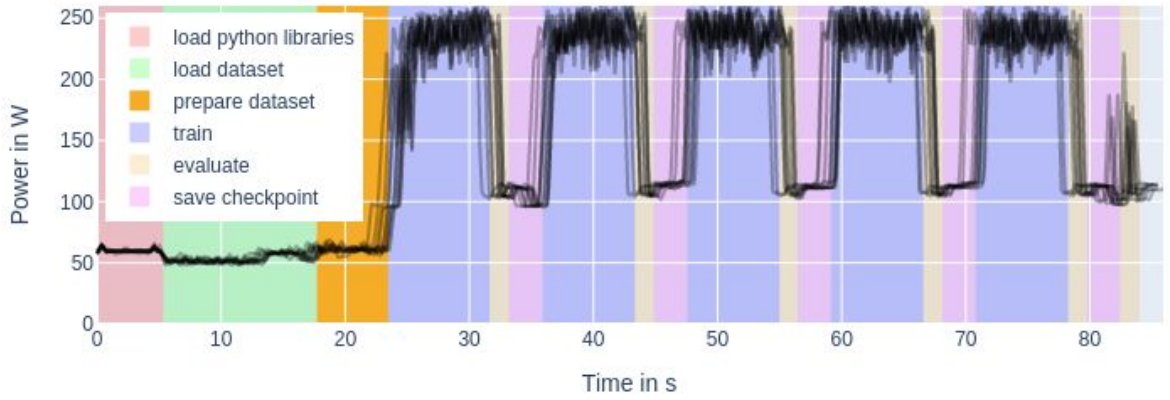
- Resume nach der 2. Epoche

- Das Programm durchläuft erneut Startphasen
- Die verbleibenden 3 Epochen werden ausgeführt



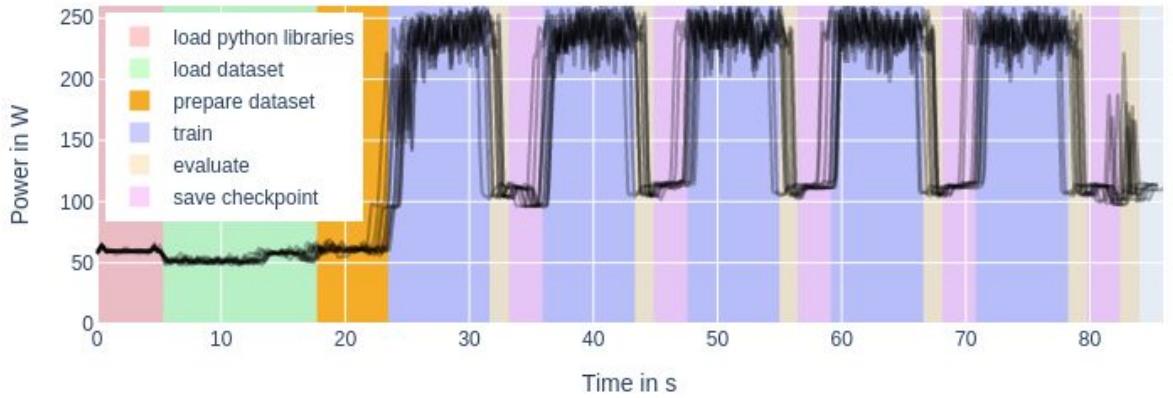
Leistungsmessungen auf Machine Learning - Zusammenfassung

- Es gibt (wiederkehrende) Phasen welche sich in ihrer Leistungsaufnahme unterscheiden
- Das Stoppen und Wiederaufnehmen von einem Job führt zu Overhead



Ein neues Workload Modell - STAWP

- Es gibt (wiederkehrende) Phasen welche sich in ihrer Leistungsaufnahme unterscheiden
- Das Stoppen und Wiederaufnehmen von einem Job führt zu Overhead




```
class Stawp(TypedDict):
    startup: List[Phase]
    work: List[Phase]

class Phase(TypedDict):
    name: str
    duration: float
    power: float
```

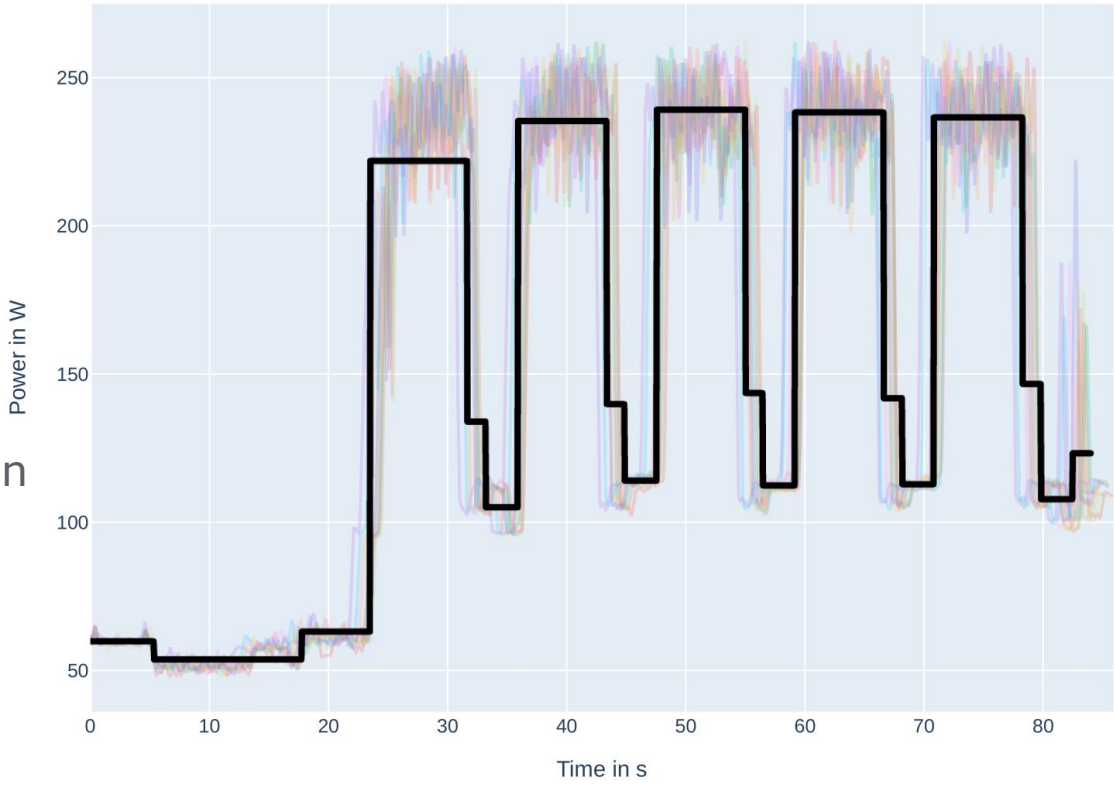
```
{
'startup': [
    {'name': 'load libraries', 'duration': 5.34, 'power': 59.9},
    {'name': 'load dataset', 'duration': 12.36, 'power': 53.77},
    {'name': 'prepare dataset', 'duration': 5.75, 'power': 63.17},
],
'work': [
    {'name': 'train, 'duration': 8.17, 'power': 221.93},
    {'name': 'evaluate', 'duration': 1.54, 'power': 134.0},
    {'name': 'save', 'duration': 2.72, 'power': 105.1},
    ...
]}
```

RoBERTa via STAWP modellieren

```
{
'startup': [
    {'name': 'load libraries', 'duration': 5.34, 'power': 59.9},
    {'name': 'load dataset', 'duration': 12.36, 'power': 53.77},
    {'name': 'prepare dataset', 'duration': 5.75, 'power': 63.17},
],
'work': [
    {'name': 'train, 'duration': 8.17, 'power': 221.93},
    {'name': 'evaluate', 'duration': 1.54, 'power': 134.0},
    {'name': 'save', 'duration': 2.72, 'power': 105.1},
    ...
]}
```

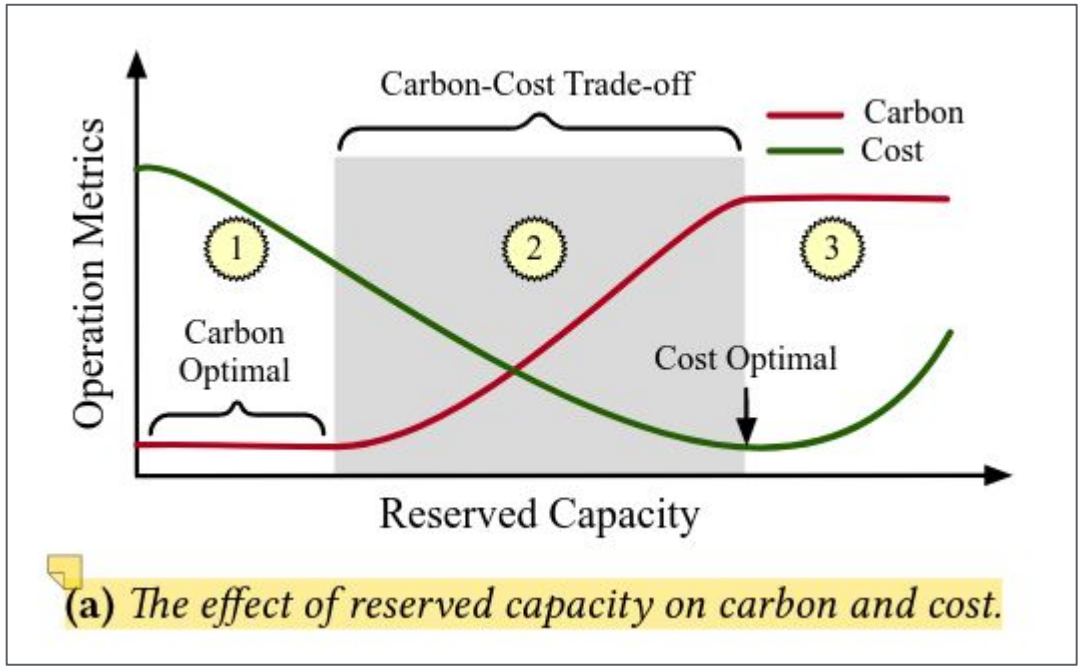
```
class Stawp(TypedDict):
    startup: List[Phase]
    work: List[Phase]

class Phase(TypedDict):
    name: str
    duration: float
    power: float
```

Treppenfunktion



Implementation

- Hanafy et al. “Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions” (Mai 2024)



(a) *The effect of reserved capacity on carbon and cost.*

- **Implementieren Simulator “GAIA” !**

Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions

Walid A. Hanafy, University of Massachusetts Amherst, USA
Qianlin Liang, University of Massachusetts Amherst, USA
Noman Bashir, Massachusetts Institute of Technology, USA
Abel Souza, University of Massachusetts Amherst, USA
David Irwin, University of Massachusetts Amherst, USA
Prashant Shenoy, University of Massachusetts Amherst, USA

Abstract

The continued exponential growth of cloud datacenter capacity has increased awareness of the carbon emissions when executing large compute-intensive workloads. To reduce carbon emissions, cloud users often temporally shift their batch workloads to periods with low carbon intensity. While such time shifting can increase job completion times due to their delayed execution, the cost savings from cloud purchase options, such as reserved instances, also decrease when users operate in a carbon-aware manner. This happens because carbon-aware adjustments change the demand pattern by periodically leaving resources idle, which creates a trade-off between carbon emissions and cost. In this paper, we present GAIA, a carbon-aware scheduler that enables users to address the three-way trade-off between carbon, performance, and cost in cloud-based batch schedulers. Our results quantify the carbon-performance-cost trade-off in cloud platforms and show that compared to existing carbon-aware scheduling policies, our proposed policies can double the amount of carbon savings per percentage increase in cost, while decreasing the performance overhead by 26%.

CCS Concepts: • **Computer systems organization** → **Cloud computing;** • **Social and professional topics** → **Sustainability**.

Keywords: Sustainable Computing, Cloud Computing

ACM Reference Format:
Walid A. Hanafy, Qianlin Liang, Noman Bashir, Abel Souza, David Irwin, and Prashant Shenoy. 2024. Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3620666.3651374

ACM ISBN 979-8-4007-0386-7/24/04.
https://doi.org/10.1145/3620666.3651374

1 Introduction

The increasing demand for computing and accelerating growth in cloud datacenter capacity have long raised concerns about their sustainability, environmental impact, and resulting carbon footprint [23]. Although cloud operators previously addressed such concerns by increasing datacenters' energy efficiency, or work done per unit of energy consumed, via software and hardware optimizations, recent work highlights that continuing improvements to energy efficiency are likely to see diminishing returns [12]. For example, large datacenters already operate near the optimal PUE value of 1.0, so there is little room to further improve PUE. Thus, to continue reducing their carbon emissions, cloud providers are increasingly employing *supply-side* optimizations, such as purchasing renewable energy from solar and wind farms to offset datacenter demand [22, 41]. However, eliminating all carbon emissions using supply-side techniques alone can be very expensive [6, 15].

Researchers have also proposed *demand-side* techniques to decrease computing's operational carbon emissions. These techniques utilize 1) visibility into grid energy's carbon intensity (in g·CO_2eq/kWh) and 2) application-level flexibility to *modulate* demand based on variations in energy's carbon intensity [6, 21, 31, 36, 44]. For example, prior work on Wait Awhile [44] and Ecovisor [35] utilize batch workloads' *temporal flexibility* to optimize carbon by executing jobs when energy's carbon intensity is low and pausing execution when carbon intensity is high. Importantly, while state-of-the-art techniques consider carbon-performance trade-offs, they ignore the cost implications of carbon-aware optimizations. Specifically, carbon-aware scheduling exploits batch jobs' temporal flexibility to delay their execution until energy's carbon intensity is low. However, this delay also increases the completion times of batch jobs. Thus, carbon-aware scheduling exhibits carbon-performance trade-off: decreasing carbon emissions generally results in longer completion times.

In addition to the performance trade-off above, there is also a cost trade-off when using temporal shifting to optimize carbon emissions. This cost trade-off manifests in

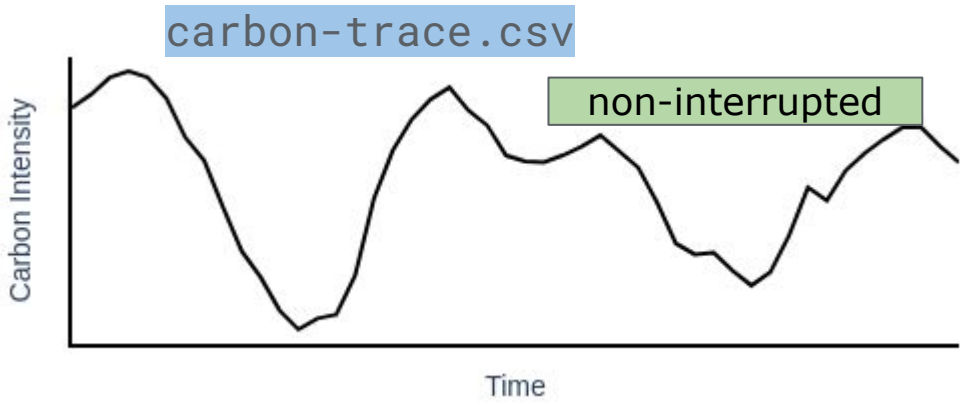
479

Implementation - GAIA bisher

```
python3 gaia/run.py
 --task-trace mytrace.csv
 --carbon-trace carbon-trace.csv
 --scheduling-policy gaia-non-interrupted
 --carbon-policy oracle
 --waiting-times 48
```

mytrace.csv

arrival_time,	length
0.0,	402166.0
419.0,	263214.0
113110.0,	166311.0
113979.0,	221466.0

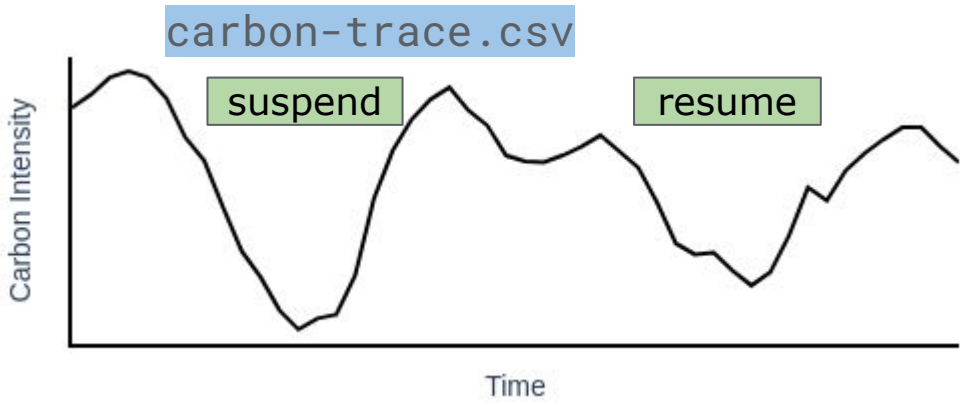


Implementation - GAIA bisher

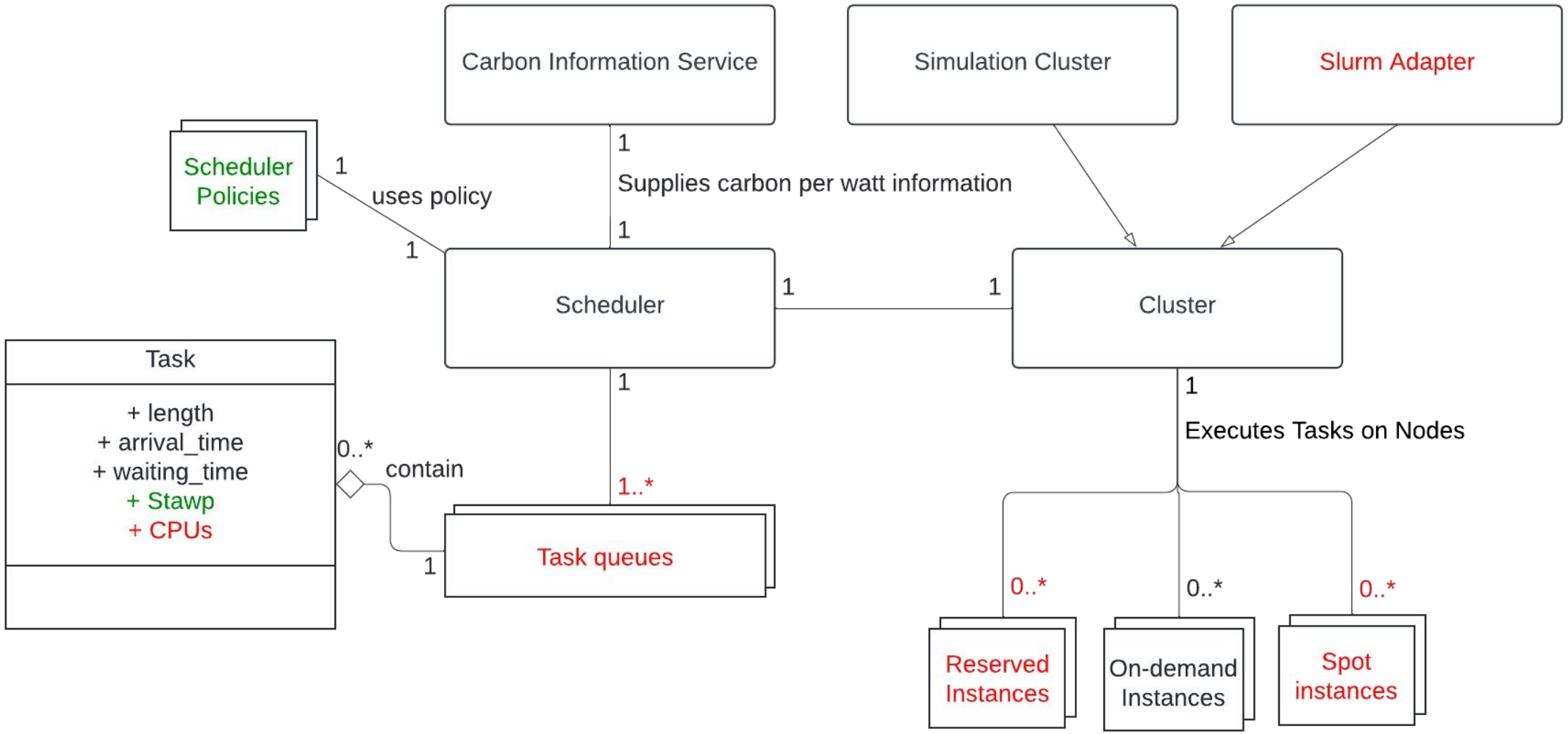
```
python3 gaia/run.py
 --task-trace mytrace.csv
 --carbon-trace carbon-trace.csv
 --scheduling-policy gaia-suspend-resume
 --carbon-policy oracle
 --waiting-times 48
```

mytrace.csv

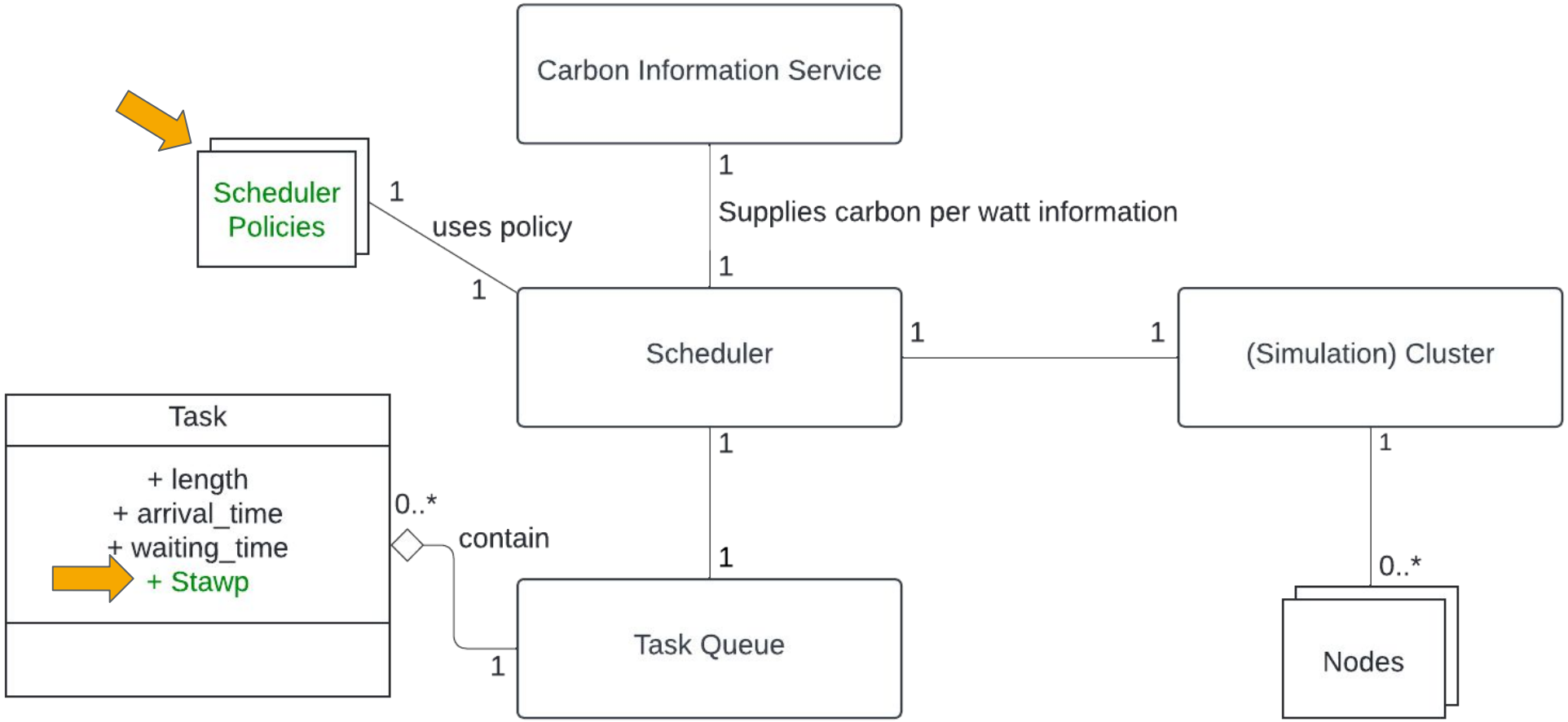
arrival_time,	length
0.0,	402166.0
419.0,	263214.0
113110.0,	166311.0
113979.0,	221466.0



Erweitern von GAIA zu meinem Testbed Carbs



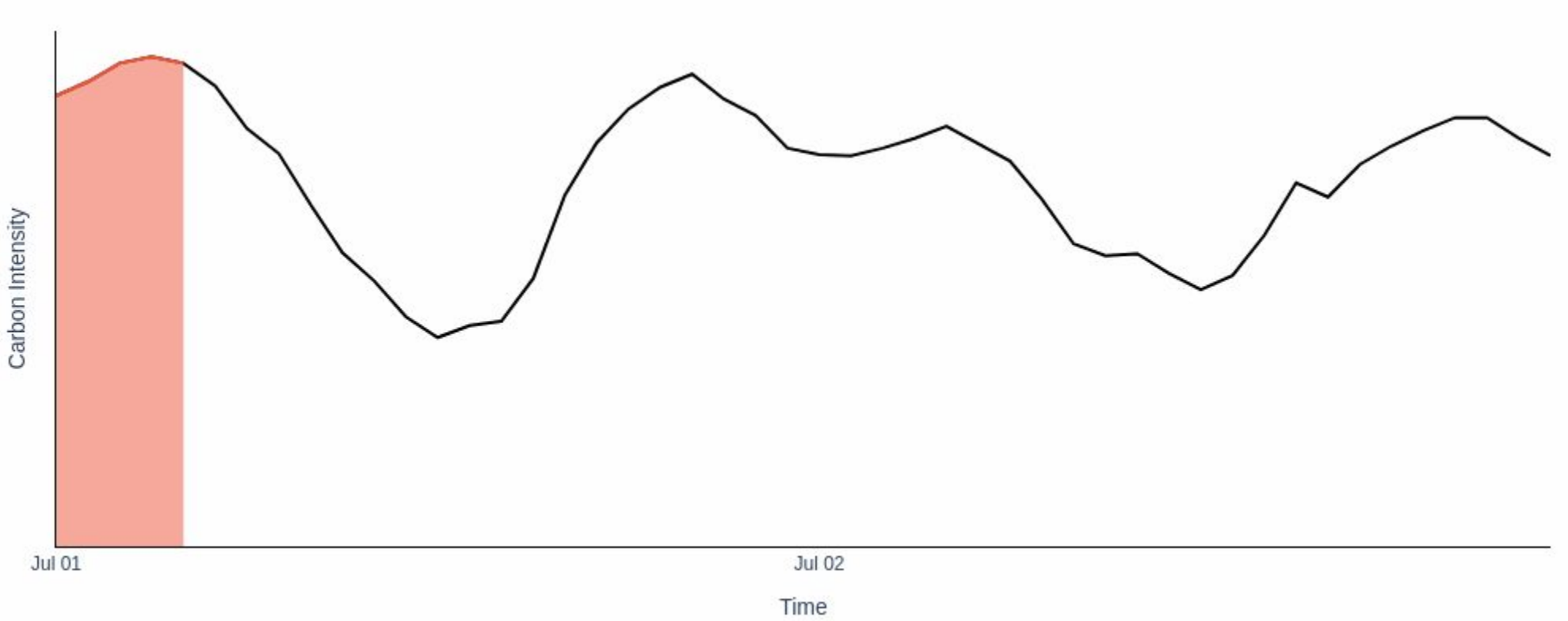
Erweitern von GAIA zu meinem Testbed Carbs



```
1.  python3 src/run.py
2.   --scheduling-policy gaia-non-interrupted
3.   --waiting-times 24
```

Carbon-Aware Scheduling, ohne Suspend, in GAIA

$$carbonEmissions = \sum_{t = start}^{start + joblength} carbonIntensity(t) \cdot power$$



```
python3 src/run.py
 --scheduling-policy carbs-non-interrupted
 --waiting-times 24
```

Carbon-Aware Scheduling, ohne Suspend, in Carbs mit STAWP

$$carbonEmissions = \sum_{t = start}^{start + joblength} carbonIntensity(t) \cdot stawp(t)$$

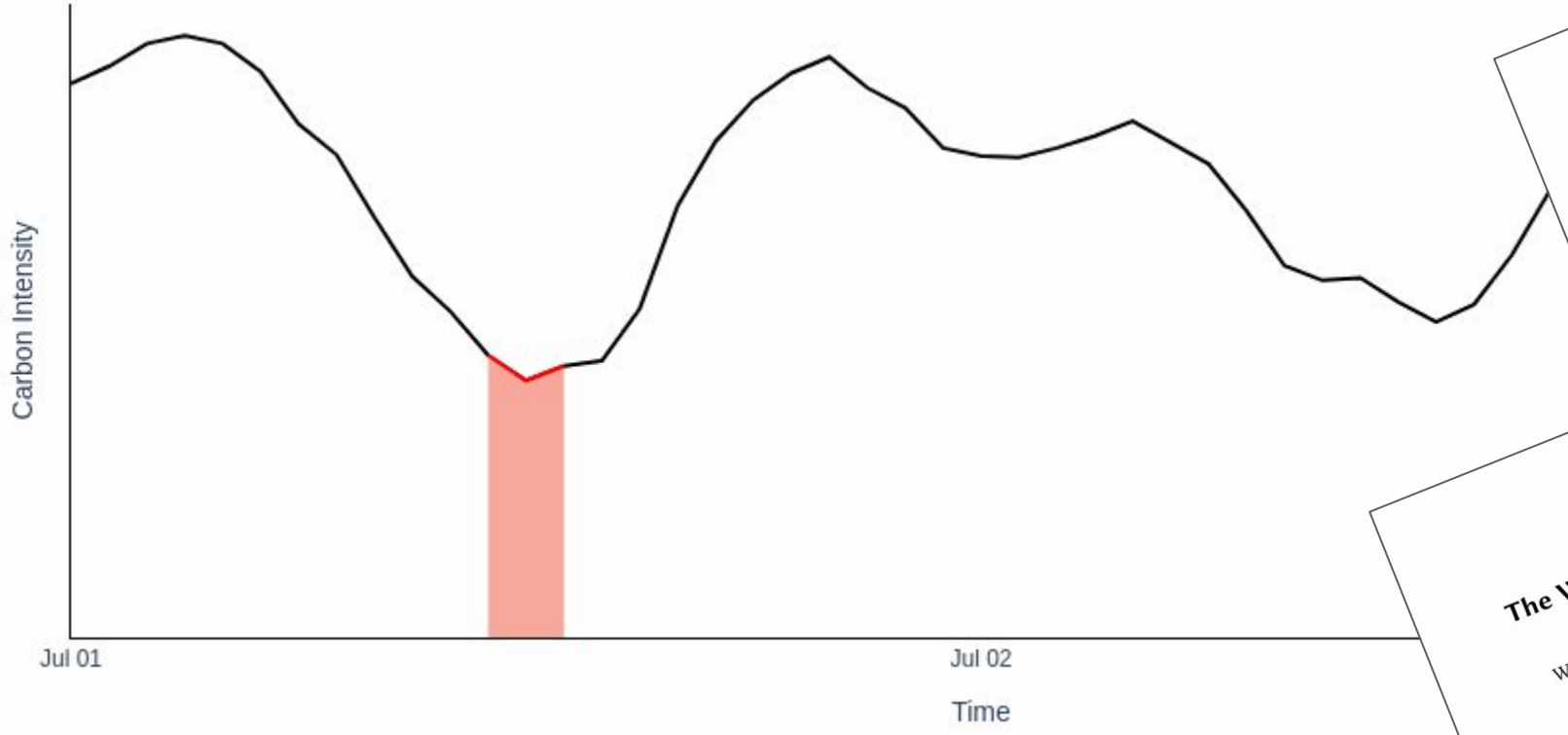
stawp(t) := Treppenfunktion aus

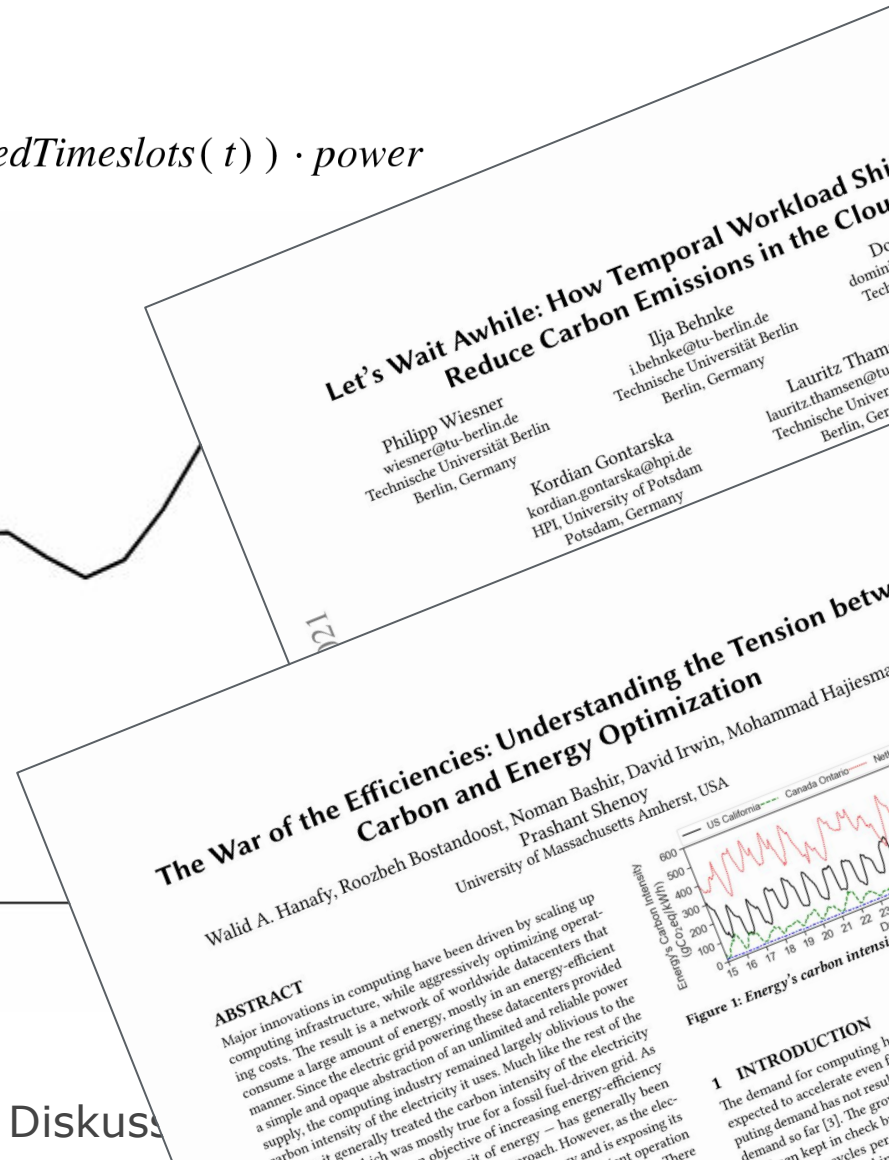
```
class Stawp(TypedDict):
    startup: List[Phase]
    work: List[Phase]

class Phase(TypedDict):
    name: str
    duration: float
    power: float
```

```
1.  python3 src/run.py
2.   --scheduling-policy carbs-suspend-resume
3.   --waiting-times 24
```

Carbon-Aware Scheduling, mit Suspend, mit GAIA

$$carbonEmissions = \sum_{t=0}^{joblength} carbonIntensity(\, sortedTimeslots(\, t)\,) \cdot power$$




Let's Wait Awhile: How Temporal Workload Shi
Reduce Carbon Emissions in the Clou
Philipp Wiesner
wiesner@tu-berlin.de
Technische Universität Berlin
Berlin, Germany
Ilja Behnke
i.behnke@tu-berlin.de
Technische Universität Berlin
Berlin, Germany
Kordian Gontarska
kordian.gontarska@hpi.de
HPI, University of Potsdam
Potsdam, Germany
The War of the Efficiencies: Understanding the Tension betw
Carbon and Energy Optimization
Walid A. Hanafy, Roozbeh Bostandoost, Noman Bashir, David Irwin, Mohammad Hajiesma
Prashant Shenoy
University of Massachusetts Amherst, USA
ABSTRACT
Major innovations in computing have been driven by scaling up computing infrastructure, while aggressively optimizing operating costs. The result is a network of worldwide datacenters that consume a large amount of energy, mostly in an energy-efficient manner. Since the electric grid powering these datacenters provided a simple and opaque abstraction of an unlimited and reliable power supply, the computing industry remained largely oblivious to the carbon intensity of the electricity it uses. Much like the rest of the
Figure 1: Energy's carbon intensi
1 INTRODUCTION
The demand for computing h
expected to accelerate even f
puting demand has not resul
demand so far [3]. The gro
kept in check b

Agenda Workload Modell Scheduler Evaluation Diskuss

Carbon-Aware Scheduling, mit Suspend, in Carbs mit STAWP

Mit Linear Programming !

Linear programs are problems that can be expressed in standard form as

$$
\begin{array}{ll}
\text{Find a vector} & \mathbf{x} \\
\text{that maximizes} & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\
\text{subject to} & A\mathbf{x} \leq \mathbf{b} \\
\text{and} & \mathbf{x} \geq \mathbf{0}.
\end{array}
$$

Here the components of $\mathbf{x}$ are the variables to be determined, $\mathbf{c}$ and $\mathbf{b}$ are given vectors, and A is a given matrix. The function whose value is to be maximized ($\mathbf{x} \mapsto \mathbf{c}^{\mathsf{T}}\mathbf{x}$ in this case) is called the objective function. The constraints $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$ specify a convex polytope over which the objective function is to be optimized.

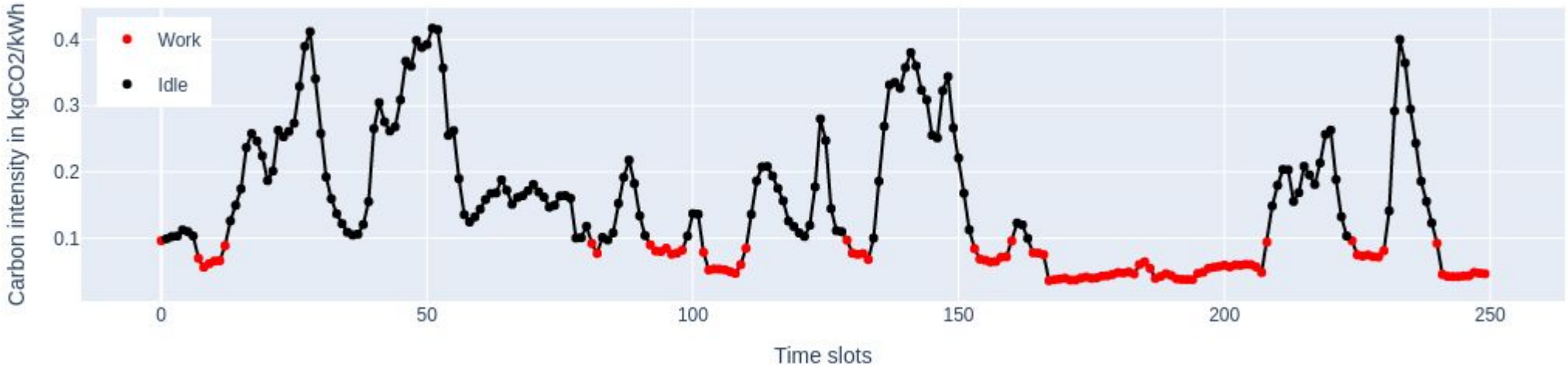
(Wikipedia)

Carbon-Aware Scheduling, mit Suspend, mit STAWP

Step 1 - den Job ausführen

```
1.  import pulp
2.  prob = LpProblem("CarbonAwareScheduling", pulp.LpMinimize)
3.  work = LpVariable.dicts("work", for t in range(DEADLINE), cat="Binary"))
4.  prob += lpSum([work[t] * carbon_cost[t] for t in range(DEADLINE)])
5.  prob += lpSum(work[t] for t in range(DEADLINE)) == WORK_LENGTH
6.  solver = pulp.Gurobi_CMD(timeLimit)
7.  prob.solve(solver)
```

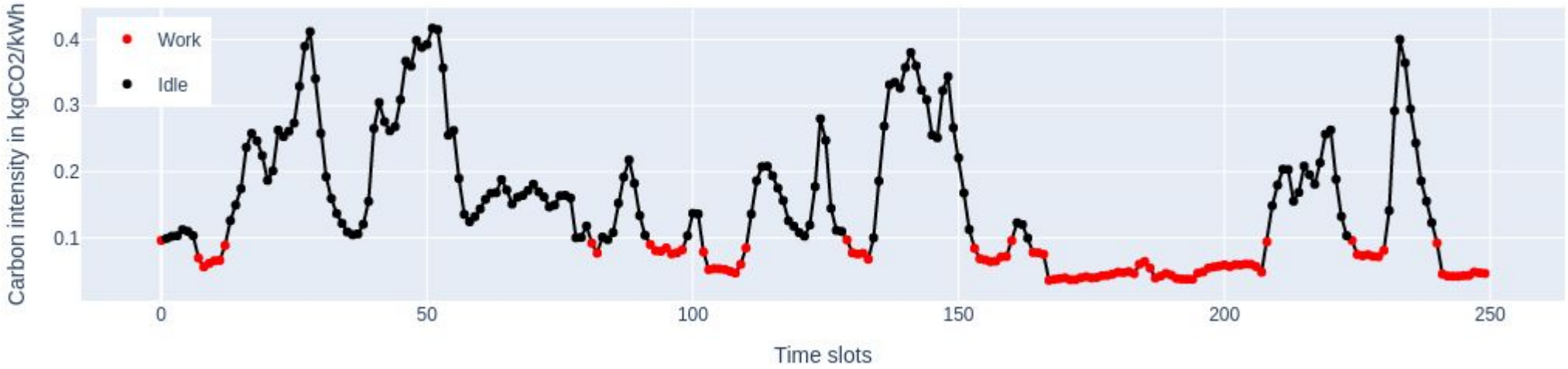
Find a vector	$\mathbf{x}$
that maximizes	$\mathbf{c}^{\mathsf{T}}\mathbf{x}$
subject to	$A\mathbf{x} \leq \mathbf{b}$



Carbon-Aware Scheduling, mit Suspend, mit STAWP

Step 2 - Overhead bei resume

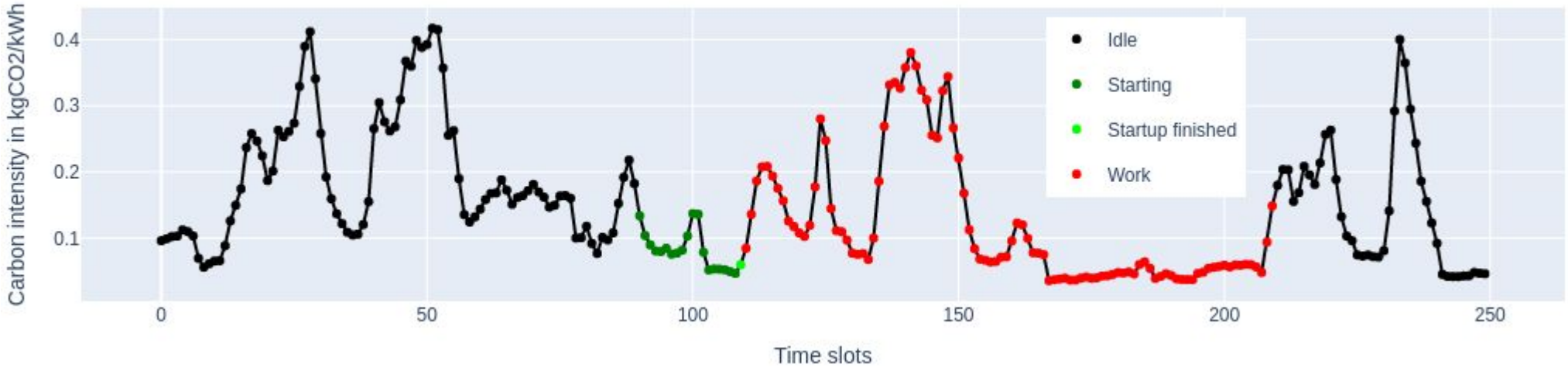
```
1.   for t in range(DEADLINE - 1):
2.    prob += startup_finished[t] >= work[t + 1] - work[t]
3.    prob += startup_finished[t] + work[t] <= 1
```



Carbon-Aware Scheduling, mit Suspend, mit STAWP

Step 2 - Overhead bei resume

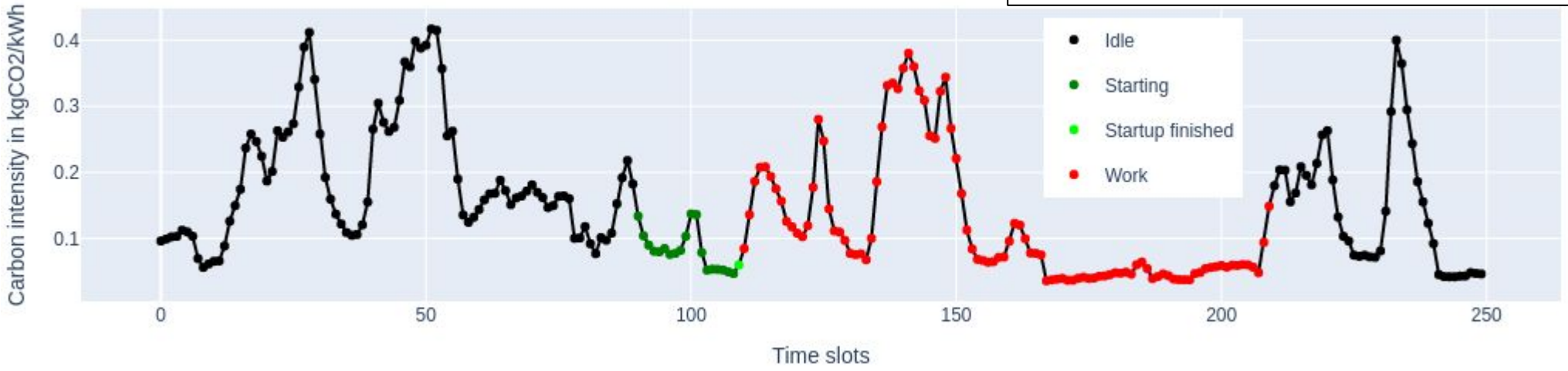
```
1.   for t in range(DEADLINE - 1):
2.    prob += startup_finished[t] >= work[t + 1] - work[t]
3.    prob += startup_finished[t] + work[t] <= 1
4.    prob += starting[t] + work[t] <= 1
5.
6.   for i in range(STARTUP_LENGTH - 1, DEADLINE):
7.    prob += pulp.lpSum([starting[i - j] for j in range(STARTUP_LENGTH)]) >= STARTUP_LENGTH * startup_finished[i]
```



Carbon-Aware Scheduling, mit Suspend, mit STAWP

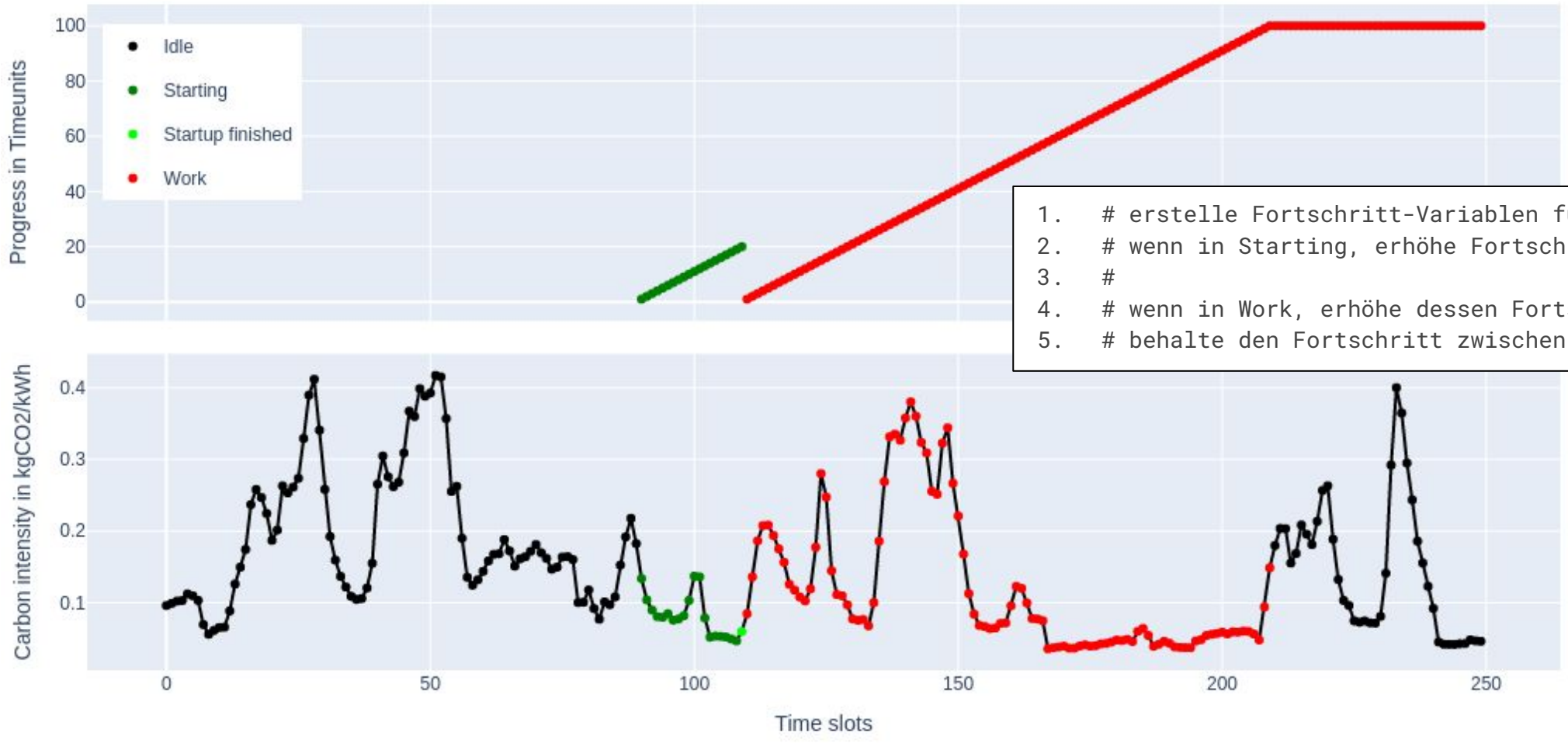
Step 3 - Zeit innerhalb des Jobs

```
1.   # erstelle Fortschritt-Variablen für Startup und Work
2.   # wenn in Starting, erhöhe Fortschritt um 1
3.   #
4.   # wenn in Work, erhöhe dessen Fortschritt um 1
5.   # behalte den Fortschritt zwischen Suspends
```



Carbon-Aware Scheduling, mit Suspend, mit STAWP

Step 3 - Zeit innerhalb des Jobs

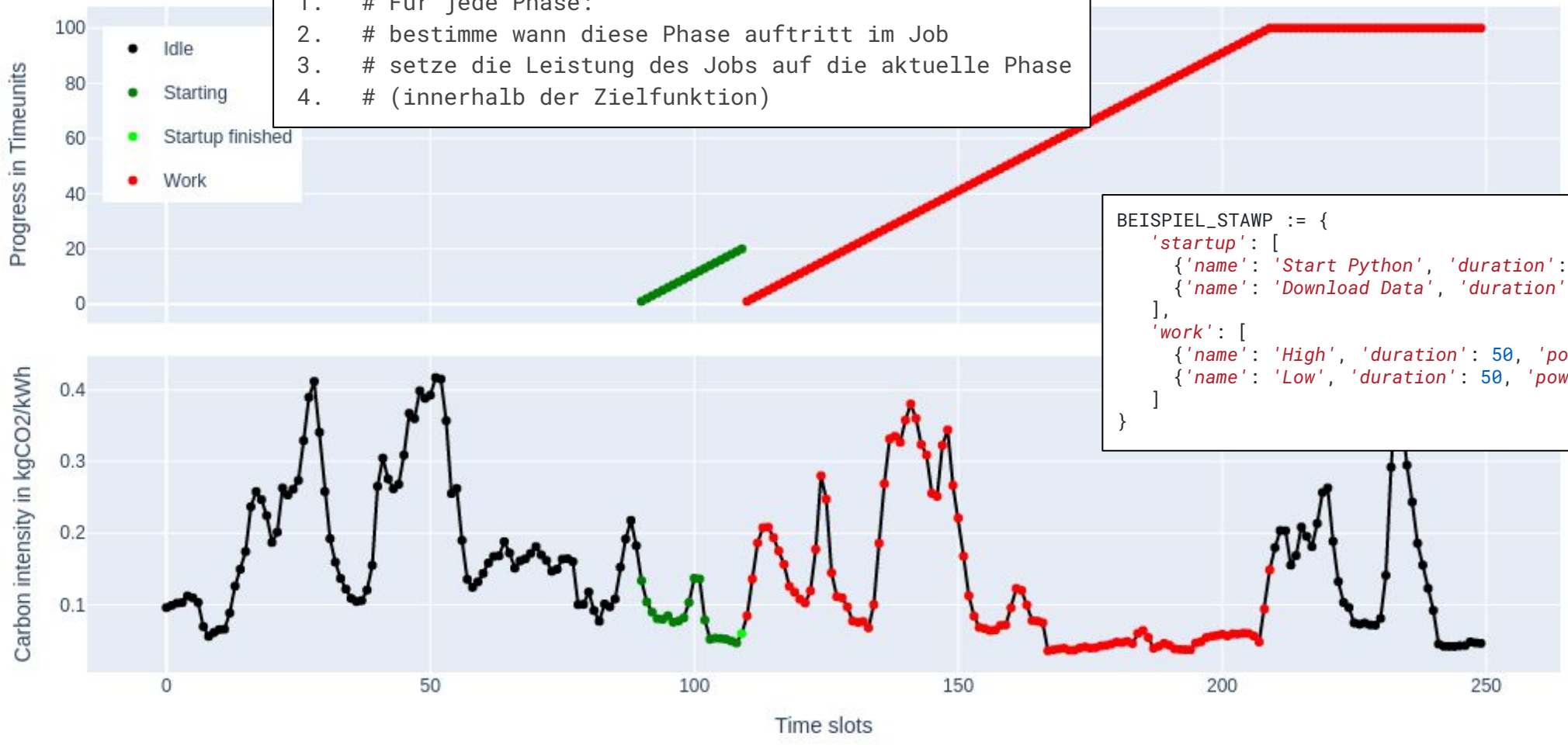



```
1.   # erstelle Fortschritt-Variablen für Startup und Work
2.   # wenn in Starting, erhöhe Fortschritt um 1
3.   #
4.   # wenn in Work, erhöhe dessen Fortschritt um 1
5.   # behalte den Fortschritt zwischen Suspends
```

Carbon-Aware Scheduling, mit Suspend, mit STAWP

Step 4 - Phasen abh. von der Zeit

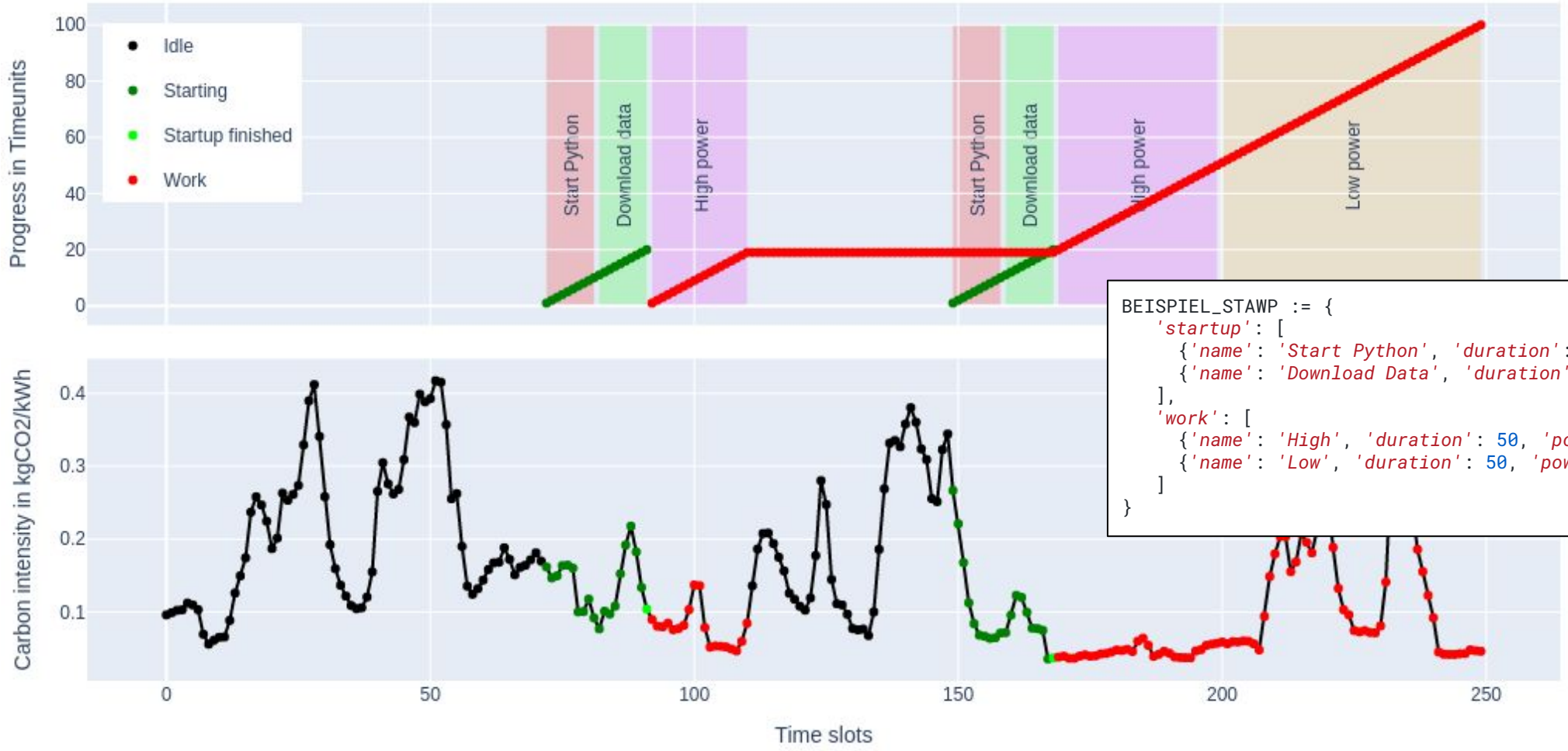
```
1.   # Für jede Phase:
2.   # bestimme wann diese Phase auftritt im Job
3.   # setze die Leistung des Jobs auf die aktuelle Phase
4.   # (innerhalb der Zielfunktion)
```




```
BEISPIEL_STAWP := {
    'startup': [
      {'name': 'Start Python', 'duration': 10, 'power': 50},
      {'name': 'Download Data', 'duration': 10, 'power': 70},
    ],
    'work': [
      {'name': 'High', 'duration': 50, 'power': 230.0},
      {'name': 'Low', 'duration': 50, 'power': 100},
    ]
}
```

Carbon-Aware Scheduling, mit Suspend, mit STAWP

Step 4 - Phasen abh. von der Zeit



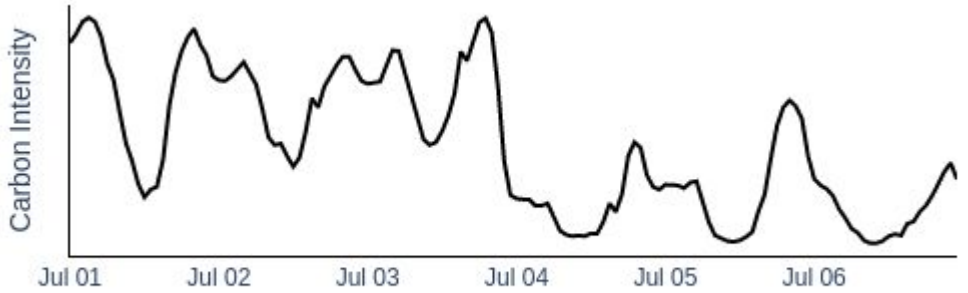

```
BEISPIEL_STAWP := {
   'startup': [
     {'name': 'Start Python', 'duration': 10, 'power': 50},
     {'name': 'Download Data', 'duration': 10, 'power': 70},
   ],
   'work': [
     {'name': 'High', 'duration': 50, 'power': 230.0},
     {'name': 'Low', 'duration': 50, 'power': 100},
   ]
}
```

Evaluation - Setup

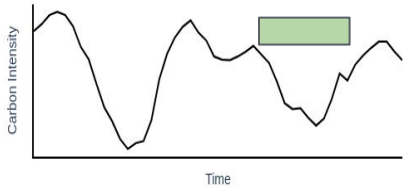
- Vergleich von STAWP gegen das bisherige State-of-the-Art Modell

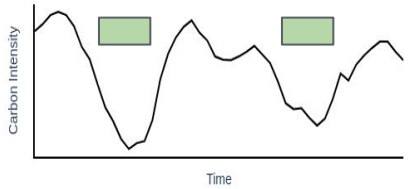
Startup Länge	kein Startup, 5, 10, und 30 minuten
Startup Leistung	100 W, 200 W
Job Länge	1, 2, 4 , 8, und 16 stunden
Phasen Länge	kurz oder lang
Benutze Phasen ?	ja oder nein

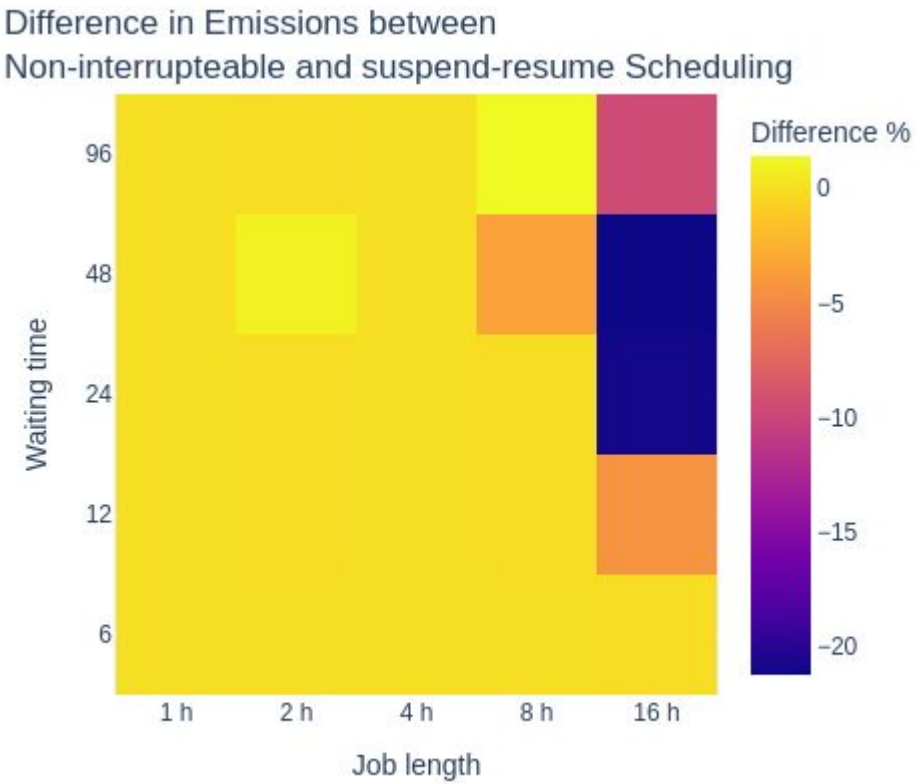
Zeitspanne	Erste Juli Woche 2024
max. Wartezeit	4h, 12h, 1, 2, und 4 Tage
Scheduler	suspend & resume, oder nicht



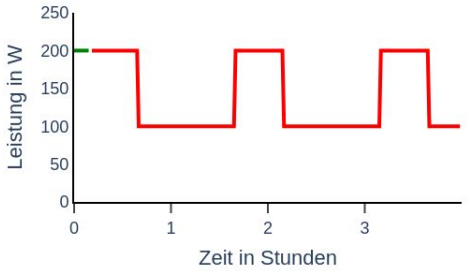
Evaluation - Vergleich von



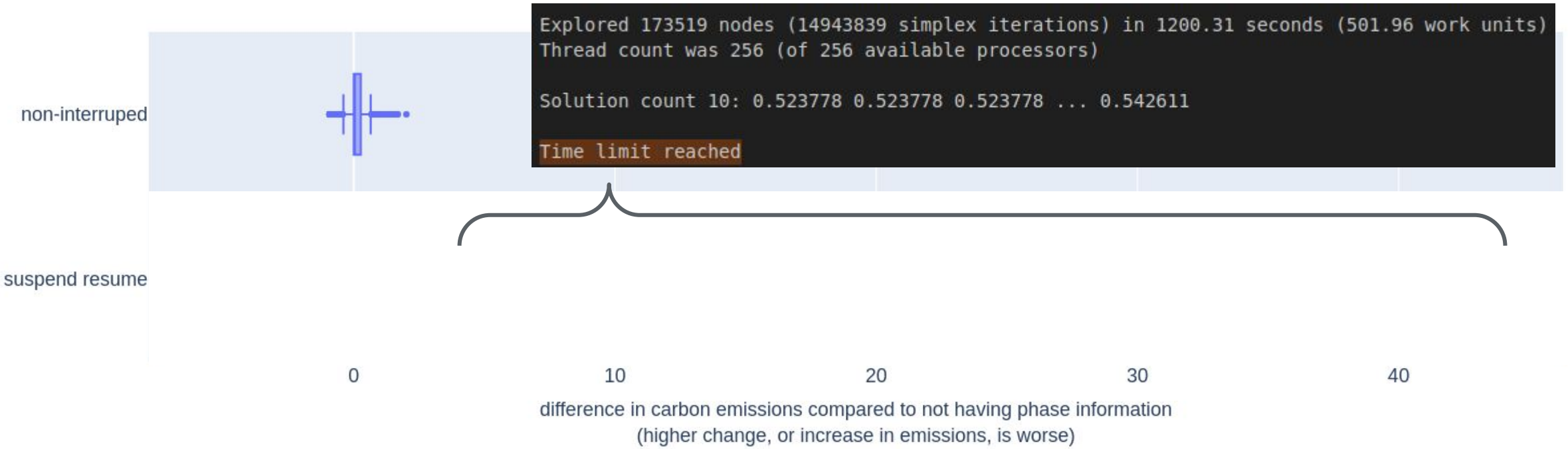




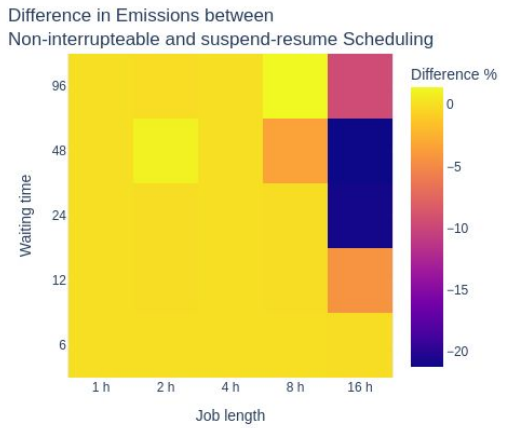
Evaluation - Vergleich von

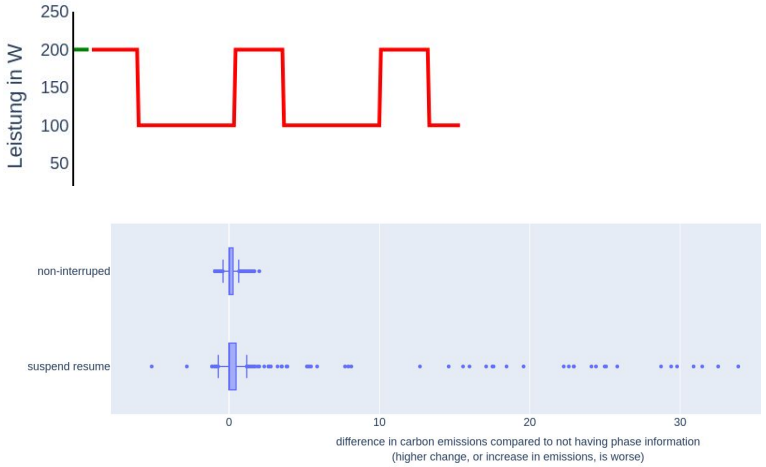


vs.



Diskussion und Future Work





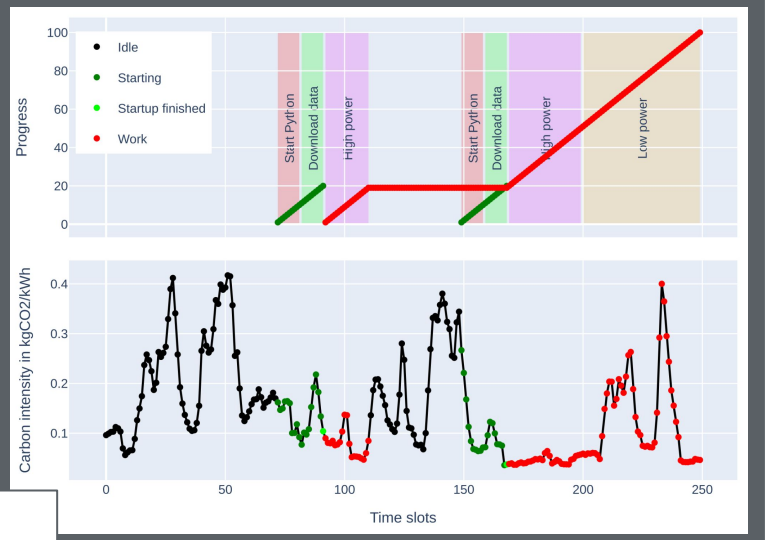

```
class Stawp(TypedDict):
    startup: List[Phase]
    work: List[Phase]

class Phase(TypedDict):
    name: str
    duration: float
    power: float
```

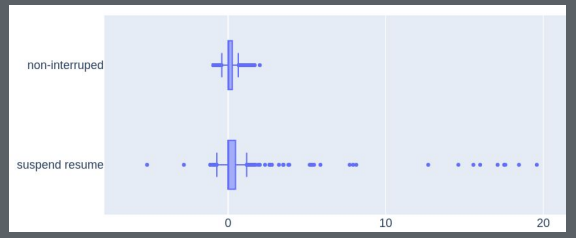
- **Beobachtungen aus Prior Work traten auch auf.**
- **Lange Jobs & Hohe Wartezeiten lohnen sich**
- **Nur bis zu 16 Stunden und auf einem Carbon Trace untersucht**

- **Effekte der Leistungsheterogenität nicht klar, Anzahl der Phasen skaliert schlecht**
- **Hier müssen andere, echte Workloads untersucht werden**

- **Annahme, dass Jobs immer suspended werden können**
- **Könnte Scheduling beeinflussen, oder für lange Jobs vernachlässigbar sein?**

```
class Stawp(TypedDict):
    startup: List[Phase]
    work: List[Phase]

class Phase(TypedDict):
    name: str
    duration: float
    power: float
```





Zusammenfassung

1. **Stawp als neues Workload Modell vorgestellt**
   enthält Startup (Overhead), Work, und Phasen
2. **Neue Scheduling Algorithmen implementiert**
   in meinem Testbed Carbs als Erweiterung von GAIA
3. **Gegen das bestehende Modell evaluiert**

Operating system	Ubuntu 24.04 LTS
Kernel	Linux 6.8.0-39-generic
CPU	AMD Ryzen 5 1600X Six-Core Processor
Memory	16GiB
GPU	GeForce GTX 1070

Table 4.1: Environment parameters of the power measurements

Listing 4.1: Used operating system information

```
MINFREQ=$(cpupower frequency-info --hwlimits | sed -n '1d;p' \
    | awk '{print $1}')
MAXFREQ=$(cpupower frequency-info --hwlimits | sed -n '1d;p' \
    | awk '{print $1}')

cpupower frequency-set --min ${MAXFREQ} &>/dev/null
cpupower frequency-set --max ${MAXFREQ} &>/dev/null

# ... conduct experiments

cpupower frequency-set --min ${MINFREQ} &>/dev/null
cpupower frequency-set --max ${MAXFREQ} &>/dev/null
```

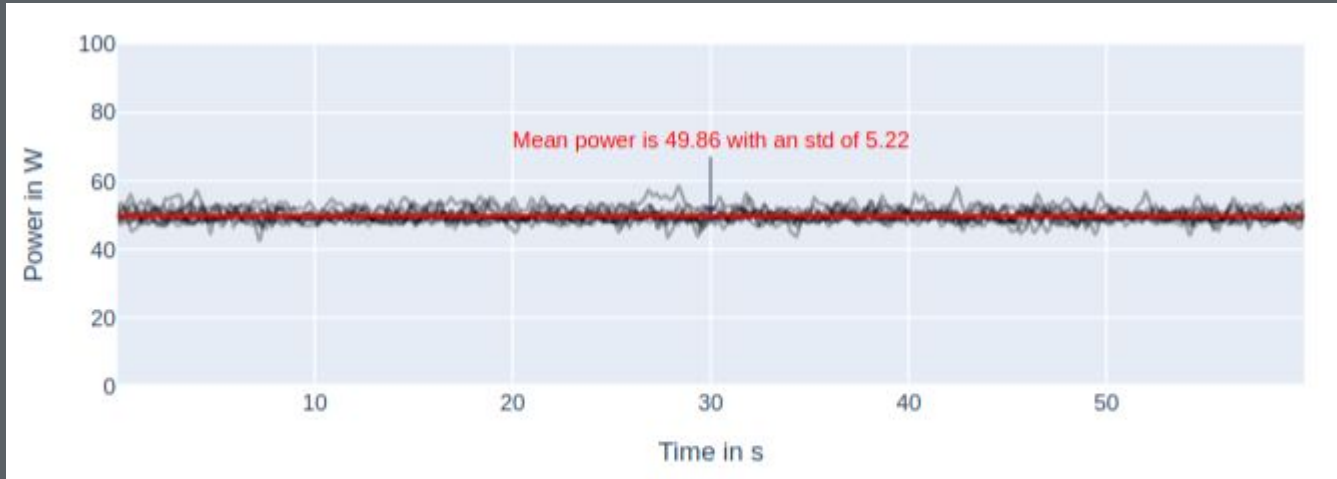


Figure 4.2: In the baseline experiment of measuring the system at rest, the average power draw is about 50 W. The black lines are Gaussian smoothed trend lines with $\sigma = 2$.

Experiment	average energy cost	standard deviation
1, whole run	12.97 kJ	0.04 kJ
2, suspend after checkpoint and resume	13.94 kJ	0.1 kJ
3, abort early and resume	15.72 kJ	0.07 kJ

Table 4.2: Energy costs of the different experiments. Unlike the no-overhead assumption in prior work, we can show that suspending a workload does carry overhead.

Listing 6.5: LP Implementation for basic scheduling

```
1 prob = LpProblem("CarbonAwareScheduling", pulp.LpMinimize)
2 work = LpVariable.dicts("work",
3     (t for t in range(DEADLINE)), cat="Binary")
4
5 prob += lpSum([work[t] * carbon_cost[t] for t in range(DEADLINE)])
6 prob += lpSum(work[t] for t in range(DEADLINE)) == WORK_LENGTH
7
8 solver = pulp.Gurobi_CMD(timeLimit=timelimit)
9 prob.solve(solver)
```

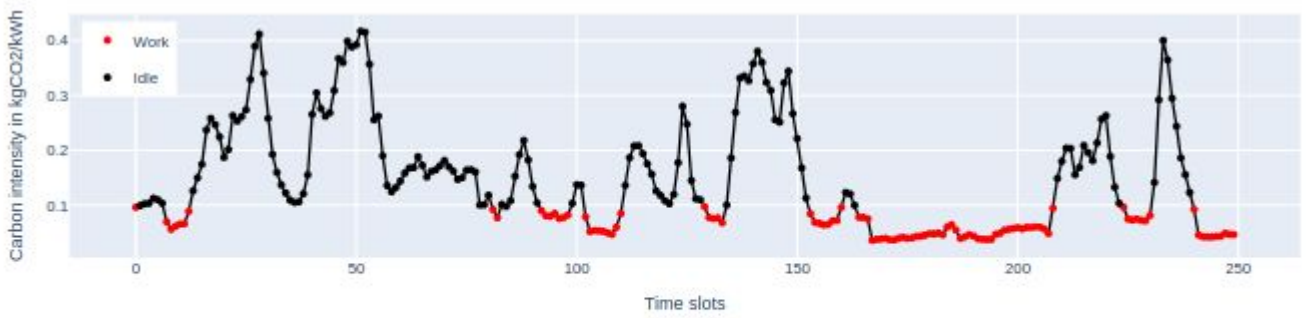


Figure 6.1: Result using the first iteration of the LP scheduler. Each time slot, with its corresponding carbon intensity, is assigned to be worked or not. As this iteration has no notion of startup-costs, only the lowest time slots are used with many suspends in between.

Listing 6.6: LP Implementation for overhead

```
1  for t in range(DEADLINE - 1):
2      prob += startup_finished[t] >= work[t + 1] - work[t]
3      prob += startup_finished[t] + work[t] <= 1
4      prob += starting[t] + work[t] <= 1
5
6  for i in range(STARTUP_LENGTH - 1, DEADLINE):
7      prob += pulp.lpSum([starting[i - j] for j in range(STARTUP_LENGTH)])
8          >= STARTUP_LENGTH * startup_finished[i]
```

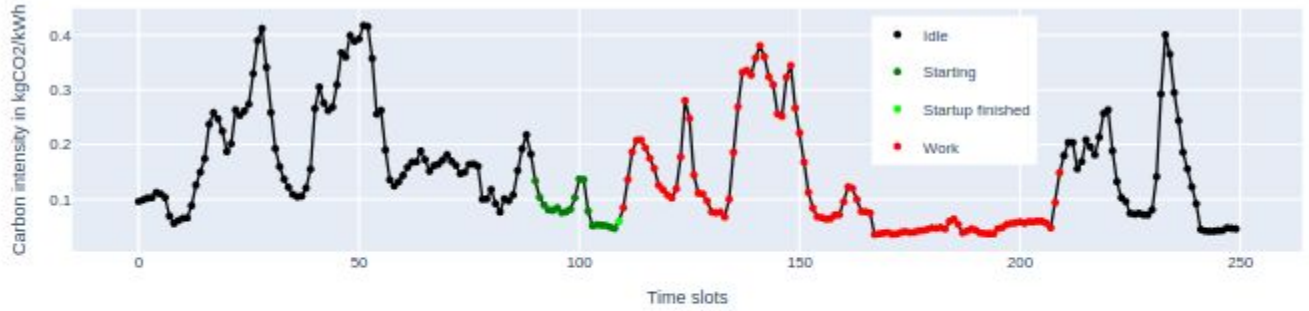


Figure 6.2: The schedule of the second iteration. In addition to time slots being assigned for work, a startup phase must also be scheduled before work can begin. The light-green mark indicates a helper variable, used for this mechanic. In comparison to the previous figure, there is now one long, connected, block where the job is worked on, as resuming a job carries an extra cost.

Listing 6.7: Progress Variables in LP

```
# define "work_progress" and "startup_progress" as
# DEADLINE-many integer variables

M = DEADLINE * 2
for t in range(DEADLINE-1):
    if t > 0:
        prob += startup_progress[t] >= startup_progress[t-1] + 1 - (1 -
            starting[t]) * M
        prob += startup_progress[t] <= startup_progress[t-1] + 1 + (1 -
            starting[t]) * M
    prob += startup_progress[t] <= starting[t] * M
    prob += work_progress[0] == 0
    if t > 0:
        prob += work_progress[t] == work_progress[t-1] + work[t]
```

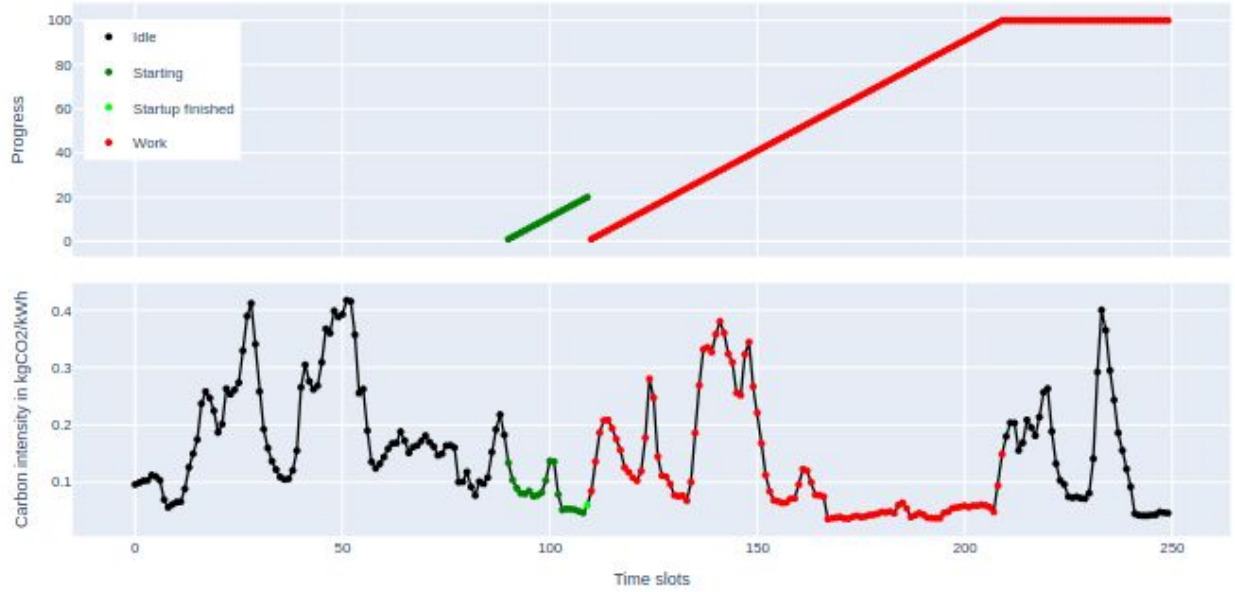


Figure 6.3: Visualisation of the result set after adding progress indicators. For either starting up or working, count up a respective integer variable. These do no yet impact the schedule itself, compared to Figure 6.2. Progress is kept even when idling.

Listing 6.8: Phase detection in LP

```
# for each phase
# let phase_indicator, phase_indicator_upper, phase_indicator_lower be
    DEADLINE-many boolean variables
# set lower_bound to be the minimum progress this phase can occur in
# set upper_bound to be the maximum similarly

# do the following for each phase
for t in range(DEADLINE):
    prob += progress[t] - lower_bound <= M*phase_indicator_lower[t]
    prob += lower_bound - progress[t] <= M*(1-phase_indicator_lower[t])

    prob += upper_bound - progress[t] <= M*phase_indicator_upper[t]
    prob += progress[t] - upper_bound <= M*(1-phase_indicator_upper[t])

    prob += phase_active[t] >= phase_indicator_lower[t] +
        phase_indicator_upper[t] - 1
    prob += phase_active[t] <= phase_indicator_lower[t]
    prob += phase_active[t] <= phase_indicator_upper[t]
```

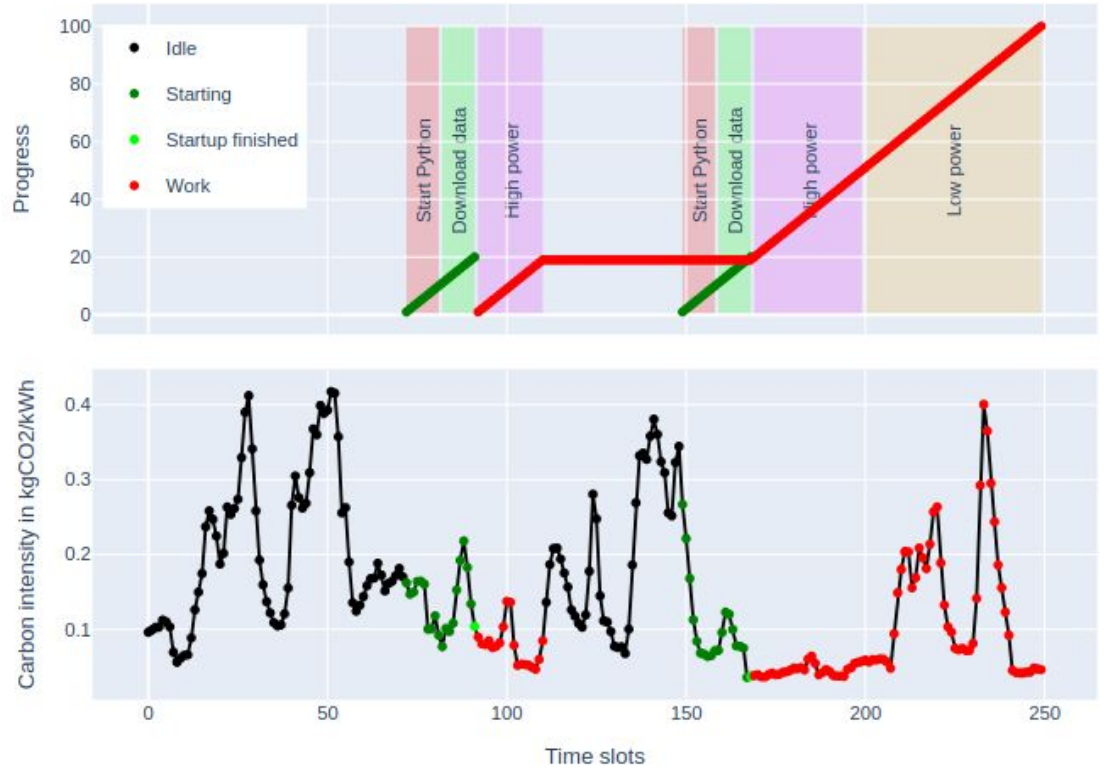


Figure 6.4: The final scheduling including differing power levels according to phases. In this example, there is a high-powered phase at 230 W and a low powered phase at 100 W. Unlike before, it is now optimal to suspend & resume the job to schedule the high-powered phase on the least carbon intensive time slots (170 to 200)

```
#!/bin/bash
export GUROBI_HOME="/home/vincent.opitz/master-thesis/gurobi1103/linux64"
export LD_LIBRARY_PATH=$GUROBI_HOME/lib
export PATH=$GUROBI_HOME/bin:$PATH
export GRB_LICENSE_FILE="/home/vincent.opitz/master-thesis/gurobi1103/gurobi.lic"
python3 src/run.py
    --scheduling-policy carbon \
    --carbon-trace DE-hourly-09-02-to-09-15 \
    --task-trace evaluation_jobs \
    --dynamic-power-draw \
    --dynamic-power-draw-type periodic-phases \
    --dynamic-power-draw-phases "{'startup':[{'name': 'startup','duration': 0, 'power': 100}],'work':[{'name':
    --carbon-policy oracle \
    --start-index 0 \
    --w 12 \
    --filename results/simulation/evaluation_jobs/carbon_periodic-phases_0_0_100_12 \
    --repeat\
```

bash Script von job-1 der Evaluation

```
arrival_time|length|cpus
0|3600.0|1
0|7200.0|1
0|14400.0|1
0|28800.0|1
0|57600.0|1
```

evaluation_jobs.csv

```
index=0

mkdir -p jobs
mkdir -p results/simulation/evaluation_jobs

rm -r jobs/job-*

# now we use a bunch of nested loops to create the cross product between these scenarios
for scheduling_policy in "${scheduling_policies[@]}"; do
    for work_type in "${work_types[@]}"; do
        for work_phase_index in "${!work_phases[@]}"; do
            for startup_length in "${startup_lengths[@]}"; do
                for startup_power_level in "${startup_power_levels[@]}"; do
                    for waiting_time in "${waiting_times[@]}"; do
                        ((index++))

                        work_phase=${work_phases[$work_phase_index]}

                        filename="results/simulation/evaluation_jobs/${scheduling_policy}_${work_type}_${work_phase_index}_${startup_length}_${st
                        phases="{'startup':[{'name': 'startup','duration': $startup_length, 'power': $startup_power_level}],'work':$work_phase}"

                        if [ ! -f "$filename" ]; then

                            echo Creating $filename

                            # create a .sh script with the job for later submission
                            echo \#!/bin/bash > jobs/job-${index}.sh
                            echo 'export GUROBI_HOME="/home/vincent.opitz/master-thesis/gurobi1103/linux64"' >> jobs/job-${index}.sh
                            echo 'export LD_LIBRARY_PATH=$GUROBI_HOME/lib' >> jobs/job-${index}.sh
                            echo 'export PATH=$GUROBI_HOME/bin:$PATH' >> jobs/job-${index}.sh
                            echo 'export GRB_LICENSE_FILE="/home/vincent.opitz/master-thesis/gurobi1103/gurobi.lic"' >> jobs/job-${index}.sh
                            echo python3 src/run.py \
                                --scheduling-policy $scheduling_policy \
```

```
work_types=("periodic-phases" "constant-from-periodic-pha
work_phases=( \
    # balanced
    "[{'name': 'high', 'power': 200, 'duration': $long},
    # long high
    "[{'name': 'high', 'power': 200, 'duration': $long},
    # short high
    "[{'name': 'high', 'power': 200, 'duration': $short}
)

startup_lengths=(0 300 600 1800)
startup_power_levels=(100 200)
waiting_times=(6 12 24 48 96)
scheduling_policies=("carbon" "suspend-resume")
```

Script für die Erstellung der Jobs der Evaluation

```bash
#!/bin/bash

chmod +x run_all_even_jobs.sh
chmod +x run_all_odd_jobs.sh

# parallelization could be better, but the gurobi license only allows two parallel session
sbatch -A polze -p magic \
    --container-image=python \
    --container-name=test-2 \
    --container-writable \
    --mem=128G \
    --cpus-per-task=128 \
    --time=24:0:0 \
    --comment="even" \
    --output=slurmlogs/output_%j.txt \
    --error=slurmlogs/error_%j.txt \
    --constraint=ARCH:X86 \
    --container-mounts=/hpi/fs00/home/vincent.opitz:/home/vincent.opitz \
    --container-workdir=/home/vincent.opitz/master-thesis/GAIA run_all_even_jobs.sh

sbatch -A polze -p magic \
    --container-image=python \
    --container-name=test-2 \
    --container-writable \
    --mem=128G \
    --cpus-per-task=128 \
    --time=24:0:0 \
    --comment="odd" \
    --output=slurmlogs/output_%j.txt \
    --error=slurmlogs/error_%j.txt \
    --constraint=ARCH:X86 \
    --container-mounts=/hpi/fs00/home/vincent.opitz:/home/vincent.opitz \
    --container-workdir=/home/vincent.opitz/master-thesis/GAIA run_all_odd_jobs.sh
```

```bash
#!/bin/bash

find jobs -name "job-*.sh" | while read -r script; do
    index=$(echo "$script" | grep -o -E '[0-9]+')

    if (( index % 2 == 0 )); then
        echo "Executing: $script"
        ./"$script"
    fi
done
```

```bash
#!/bin/bash

find jobs -name "job-*.sh" | while read -r script; do
    index=$(echo "$script" | grep -o -E '[0-9]+')

    if (( index % 2 == 1 )); then
        echo "Executing: $script"
        ./"$script"
    fi
done
```

Submitten aller Jobs in Slurm

Listing 7.1: Submitting the evaluation to SCORE Lab's Slurm

```
sbatch -A polze -p magic \
    --container-image=python \
    --container-name=test \
    --container-writable \
    --mem=128G \
    --cpus-per-task=128 \
    --time=24:0:0 \
    --output=slurmlogs/output_%j.txt \
    --error=slurmlogs/error_%j.txt \
    --constraint=ARCH:X86 \
    --container-mounts=/hpi/fs00/home/vincent.opitz:/home/vincent.opitz \
    --container-workdir=/home/vincent.opitz/master-thesis/carbs \
    run_all_even_jobs.sh
```

Parameter	p value
Job length	0.00
Waiting time	0.00
Phases	0.0001
Scheduling Algorithm	0.18
Startup power level	0.66
Phases averaged or not	0.74
Startup length	0.99

Table 7.2: Results of the ANOVA for total carbon emissions per scheduler. The lower the p-value, the higher the impact on carbon emissions.

carbs > slurmlogs > output_1215025.txt

```
501   H280997 76721                      0.5237778    0.43812  16.4%   214 1084s
503    287337 76914     cutoff   56         0.52378    0.44016  16.0%   212 1091s
504    292331 76475    0.48840   59  435    0.52378    0.44121  15.8%   210 1095s
505    305445 74293    0.51606   63  364    0.52378    0.44432  15.2%   205 1102s
506    311380 73262    0.51210   55  592    0.52378    0.44604  14.8%   203 1107s
507    318109 72136    0.51135   61  520    0.52378    0.44797  14.5%   201 1110s
508    331886 69693    0.47927   52  832    0.52378    0.45052  14.0%   197 1119s
509    338987 68893     cutoff   59         0.52378    0.45179  13.7%   195 1126s
510    350442 65975     cutoff   56         0.52378    0.45433  13.3%   192 1131s
511    358194 64206     cutoff   54         0.52378    0.45554  13.0%   190 1136s
512    366285 62221    0.50677   55  589    0.52378    0.45736  12.7%   188 1141s
513    375823 59977     cutoff   60         0.52378    0.45799  12.6%   186 1146s
514    384769 57922     cutoff   61         0.52378    0.45990  12.2%   184 1151s
515    393709 55782     cutoff   60         0.52378    0.46131  11.9%   182 1156s
516    402240 53577     cutoff   58         0.52378    0.46275  11.7%   180 1162s
517    412712 53450     cutoff   57         0.52378    0.46387  11.4%   178 1168s
518    422946 54350    0.51404   54  423    0.52378    0.46437  11.3%   176 1177s
519    437245 54955     cutoff   57         0.52378    0.46712  10.8%   174 1184s
520    448501 55311     cutoff   59         0.52378    0.46861  10.5%   172 1190s
521    460202 55356    0.50235   64  409    0.52378    0.46962  10.3%   171 1197s
522    472395 55240    0.50530   56  500    0.52378    0.47130  10.0%   169 1200s
523
524   Cutting planes:
525     Learned: 478
526     Gomory: 22
527     Cover: 8048
528     Implied bound: 80
529     Clique: 3376
530     MIR: 423
531     StrongCG: 73
532     Flow cover: 1515
533     Inf proof: 13
534     Zero half: 647
535     Mod-K: 2
536     Network: 8
537     RLT: 262
538     Relax-and-lift: 610
539     BQP: 20
540
541   Explored 476681 nodes (80360387 simplex iterations) in 1200.84 seconds (767.94 work units)
542   Thread count was 256 (of 256 available processors)
543
544   Solution count 10: 0.523778 0.523778 0.523778 ... 0.573111
545
546   Time limit reached
547   Best objective 5.237777777782e-01, best bound 4.712998343346e-01, gap 10.0191%
548
```

Ausschnitt der Evaluations Logs

```
1    gap
2    0.0000
3    0.0000
4    0.0000
5    10.0191
6    91.3230
7    0.0000
8    0.0000
9    0.0000
10   0.0000
11   0.0000
12   0.0000
13   0.0000
14   0.0000
15   0.0000
16   0.0000
```

Filtern der Logs nach der Gap Value
$ logs | grep “Best Object” | awk …

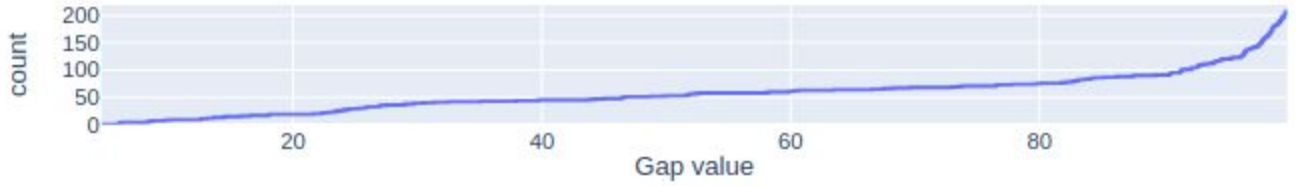
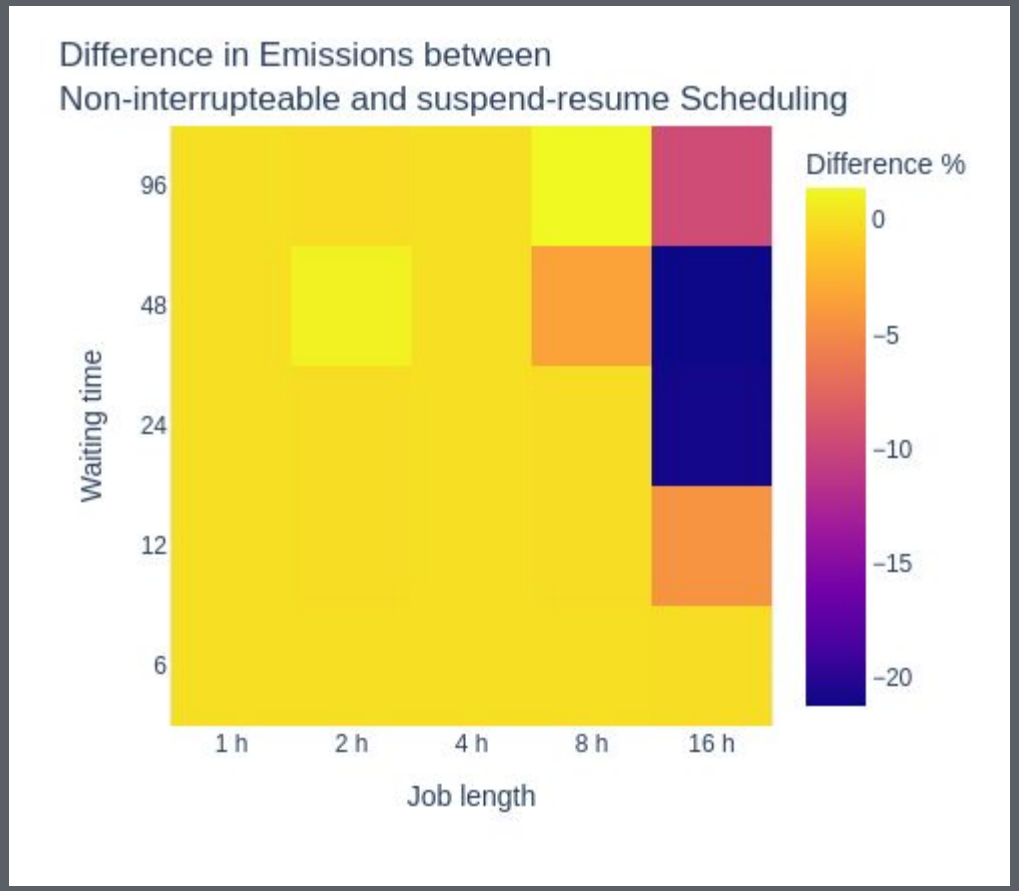
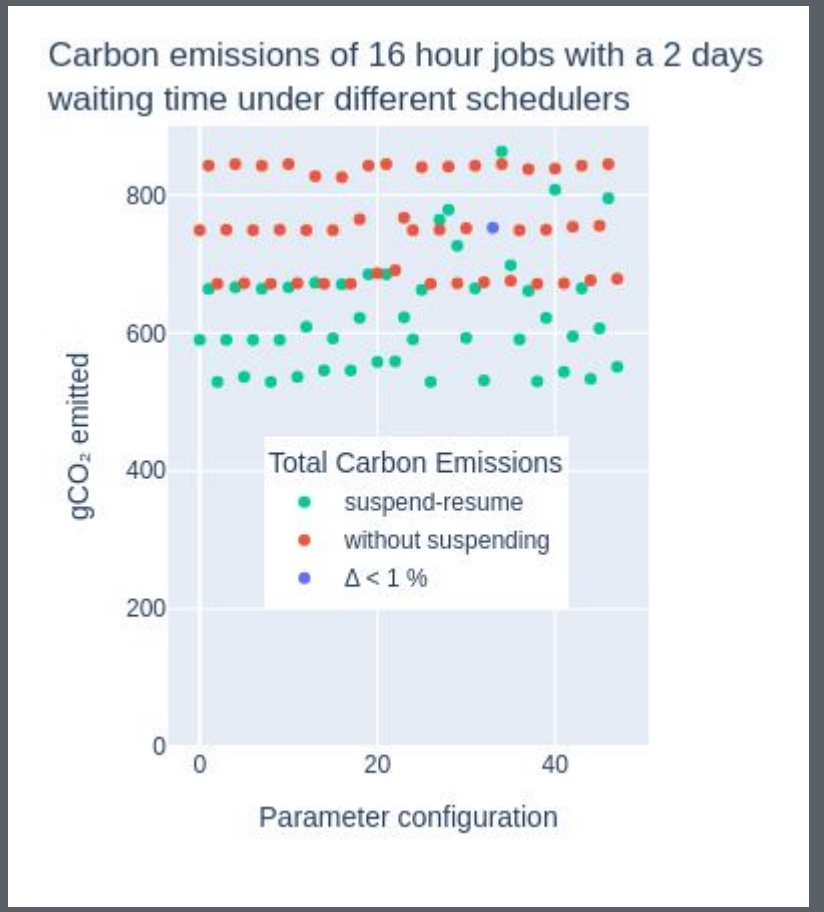
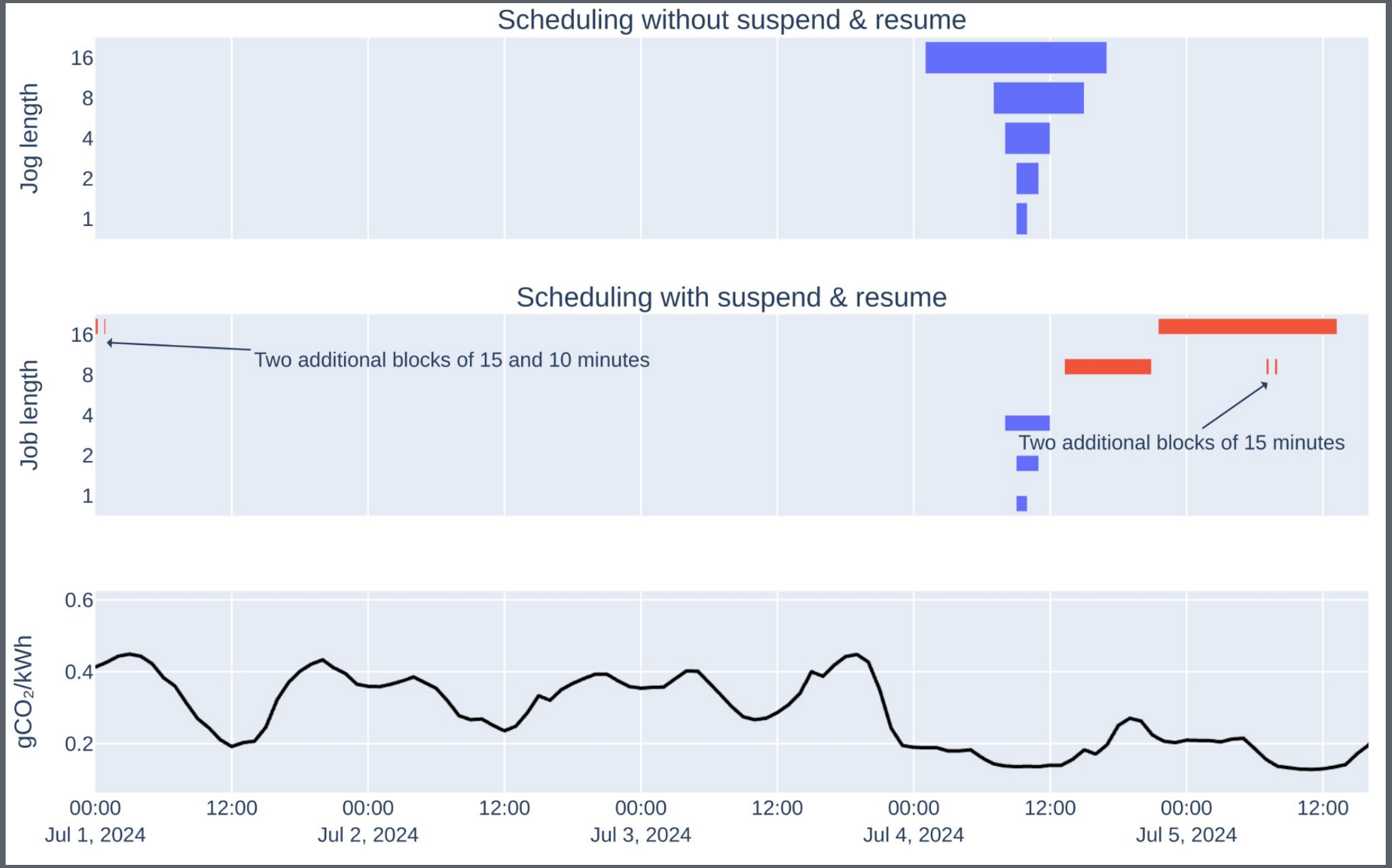


Figure 7.3: Cumulative distribution plot of the 209 gap values of time limited schedules. A higher gap indicates worse results. There is a heavy skew to 100%, with half of the values being above 95%

Figure 7.5: Normalized carbon emissions with the baseline being a wait time of 6 hours for each constant-power job. There are small difference for jobs with a startup length of 30 minutes but for the most part, there is no difference across the remaining parameters.


Difference in Emissions between
Non-interrupteable and suspend-resume Scheduling
Waiting time
96
48
24
12
6
1 h
2 h
4 h
8 h
16 h
Job length
Difference %
0
−5
−10
−15
−20


Carbon emissions of 16 hour jobs with a 2 days
waiting time under different schedulers
gCO₂ emitted
800
600
400
200
0
0
20
40
Parameter configuration
Total Carbon Emissions
suspend-resume
without suspending
Δ < 1 %


Scheduling without suspend & resume
Jog length
16
8
4
2
1
Scheduling with suspend & resume
Job length
16
8
4
2
1
Two additional blocks of 15 and 10 minutes
Two additional blocks of 15 minutes
gCO2/kWh
0.6
0.4
0.2
00:00
Jul 1, 2024
12:00
00:00
Jul 2, 2024
12:00
00:00
Jul 3, 2024
12:00
00:00
Jul 4, 2024
12:00
00:00
Jul 5, 2024
12:00