A parameterized test bed for carbon aware job scheduling

Ein parametrisierbares Testbed für kohlenstoffbewusste Jobplanung

Vincent Opitz

Arbeit zur Erlangung des Grades “Master of Science” der
Digital-Engineering-Fakultät der Universität Potsdam

A parameterized test bed for carbon aware job scheduling

Ein parametrisierbares Testbed für kohlenstoffbewusste Jobplanung

Vincent Opitz

Arbeit zur Erlangung des Grades “Master of Science” der Digital-Engineering-Fakultät der Universität Potsdam

A parameterized test bed for carbon aware job scheduling
(Ein parametrisierbares Testbed für kohlenstoffbewusste Jobplanung)
von Vincent Opitz

Arbeit zur Erlangung des Grades “Master of Science” der Digital-Engineering-Fakultät der Universität Potsdam

Betreuer:in: Prof. Dr. rer. nat. habil. Andreas Polze
Universität Potsdam,
Digital Engineering-Fakultät,
Fachgebiet für Betriebssysteme und Middleware

Gutachter:in: Prof. Dr. Jack Alsohere
University of San Serife
Facultiy of Computer Doings

Amelia van der Beenherelong
ACME Cooperation

Datum der Einreichung: 9. September 2024

Zusammenfassung

ummarize thesis

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

Zusammenfassung

S

ummarize thesis

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1 Introduction

- I could start with a fact stating that datacenters use X amount of energy which in turn emits carbon for its production
- We can reduce the amount of carbon emitted for the same amount of work by scheduling our work in smart way
- this is not a new idea; datacenters around the world are already using data provided by e.g. electricitymaps
- one thing that is not represented in literature however is the heterogenity of jobs, as well as the emphcheckpoint & resume capabilities of some jobs, particularly machine learning jobs which are projected to make UP X amount of energy in the coming years [QUOTATION NEEDED]
- I want to create a better model for jobs in datacenters
- I then want to use that new model to investigate wether we can exploit the heterogenity of jobs to further reduce carbon emissions
- One hypothesis is for example that, we can match high-power phases of jobs to the lowest carbon-intensive time frames, so we do not "waste" those rare times with low-power phases
- We can also investigate wether the previously proposed stop & resume techniques still work if the assumption that stopping and resuming a job has no overhead is broken.
- We hypothesis that a stop & resume strategy can stil lead to reduced carbon emissions but only on certain jobs.

2 Background

This chapter will be used to introduce some basic terminology as well as some basic arguments on why carbon-aware scheduling is useful.

The composition of the public grid The public grid is made up of many energy producers. This *energy mix* is composed of different sources: low-carbon technologies like solar, wind, or hydroelectricity and carbon-intensive sources like coal, gas, or oil.

Figure 2.1 shows one such energy example, for a specific location, in this case Germany. As it is summer, solar production makes up large amount of power at that time. Germany is further interesting in the sense, that it is a country that has no nuclear power stations anymore.

Over the day, the demand and supply for power changes. Carbon-efficient sources will generally follow the weather: Solar production follows the amount of sun shining and thus follows a *diurnal* rhythm over the day. Wind and hydroelectricity will also depend on the weather. Nuclear energy is generally also considered carbon-efficient, but is generally not useful for carbon-efficient scheduling as the amount of energy produced from is generally stable throughout the day, meaning that there is little use in deferring work to a later time.

QUOTATION NEEDED, maybe use look in on of the related work papers

As near-time weather predictions have high accuracy, this enables X.

Thus, at each point in time the amount of CO_2 per unit of electric power can be determined by averaging the amount of power each source supplies to the grid and how much carbon it emits. Figure 2.2 is an example X-day timeframe of the carbon-curve.

QUOTATION NEEDED

Power grid Signals Carbon-aware scheduling commonly works two possible metrics or signals: the *average emissions* are the metric describing the amount of carbon per unit of energy, basically what has been used in the text so far. Another metric is the *marginal emissions*, answering the question of "if more energy is used at this point in time, how much carbon would that cost?". The answer to that question is usually that non-renewable power plants have to increase production as renewable sources cannot increase production to meet demand (as the sun and wind intensity are set by the weather). Going by the marginal metric, any kind of carbon aware scheduling would be pointless, as any work at any point in time would result in the same increase of carbon. Thankfully, there are secondary reasons that would render carbon-aware scheduling useful. One of them are is *curtailment*. This encompasses any methods that reduce the amount of produced renewable energy. As the power grid always needs to have a balance between demand and production, curtailment methods such as turning of wind turbines, selling power at a loss, or charging batteries may be used. Carbon aware scheduling via the average emissions signal, would lower the amount of curtailment needed, as demand for energy would increase at those times. Another argument can also be made that by increasing

Find the quote

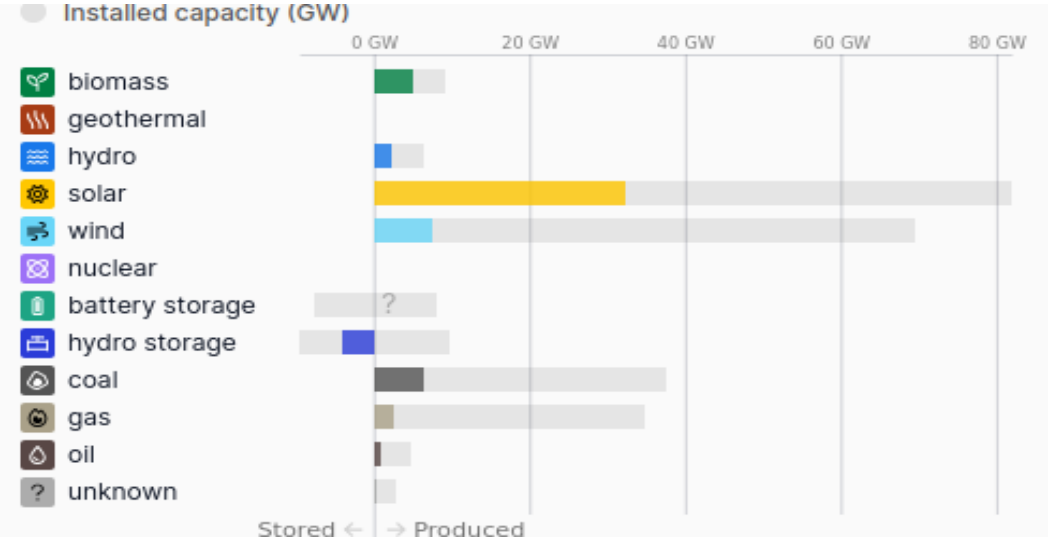


Figure 2.1: An example snapshot of the energy mix in Germany, at DATE

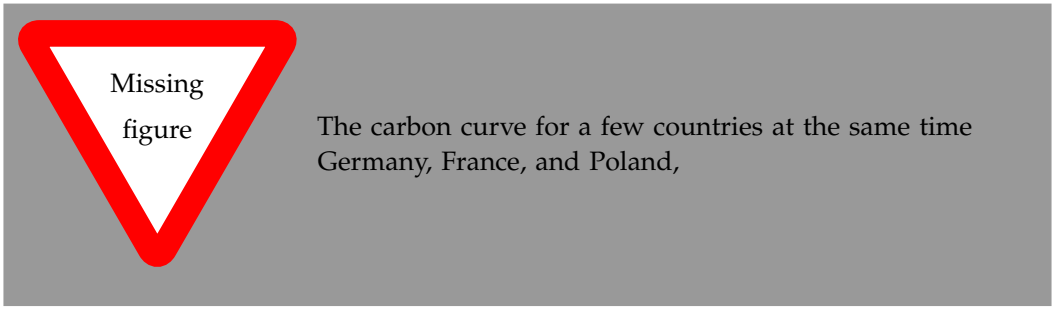


Figure 2.2: Carbon in per unit of energy in select countries

power demand during times when renewable production is high, energy producers would be "signalled" that there is demand for renewable energy. Lastly, as renewable energy is generally cheaper in production than non-renewable energy, scheduling work on low carbon-periods coincides with cheaper energy prices as well. While this would not reduce costs on most energy contracts, there are also some contracts that use dynamic pricing[1]

Need a citation here!

Types of work in a datacenter According to [1], there are 3 environments in which scheduling may take place. *Batch systems* describe environments in which there is no user interaction. A user may submit their job, and it would be executed according to the scheduler at some point in time. On the contrary, in *interactive* settings, a user would be interacting with the system "live", and would thus expect quick responses to their inputs. The last environment would be a *real time* system. There deadlines and predictability would dictate how a scheduler would operate.

Does this need more examples?

For this work's topic, carbon-aware scheduling, only batch systems will be looked at as they allow more freedom to the times of scheduled jobs.

Which metrics are there POI-A, etc.

I could also add more stuff about metrics and so on, but I am not using them yet in my implementation

Power intake of a computer As there will be power measurements in 4.1.1, some basic understanding of energy and power used for computation will be provided:

- I could mostly borrow from the EBRH slides; there is some base power needed that is correlated to the hardware (dynamic and static energy)
- this also depends on frequency (which is why later we set our CPU frequency to some hard coded value [or do not do that in the case of my GPU lol])
- basically, use this paragraph to outline all things we take care of in my power measurements

[1] One example for dynamic pricing would be *Tibber*: `https://tibber.com/de`

3 Related Work

A systematic approach I used the following system for finding related work to my topic; first, my supervisors and I would brainstorm for search engine keywords. The specific words are in the attachments, but can be grouped into two groups: one for all things carbon-aware and one for keywords about servers / computing / HPC and such.

Literaturrecherche aufarbeiten und hier verlinken

Using these two groups, I could then create a Google Scholar queries via the cross product between them. Using the double-quotation feature would further limit the results. For each query, I would then read the abstracts of roughly the first 5 results, depending on if their titles sounded subjectively fit.

These would then be entered into a spreadsheet: for each query, 5 paper titles. Additionally, I would also further explore papers through *connected papers*[2] or looking up individual authors on a subjective basis. The papers from the 2023 and 2024 HotCarbon Workshops[3] also proved to be related.

I then "rated" each paper into three categories:

- Green, meaning that they seem very connected and are good first entries into the topic
- Orange, which would indicate that are somehow connected to the paper and might be read at a later date
- Red, the paper is either irrelevant or had some other flaw. These would not be touched again in the course of my work

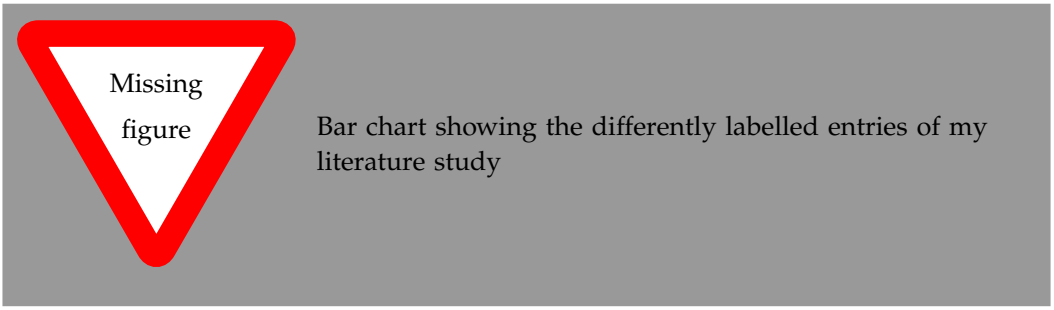
With this approach, X abstracts were read. Figure 3.1 shows that X abstracts were deemed "good" for the purpose of this work. The complete list of papers analyzed can be found in

Figure out results of the literature stuff

THE ATTACHMENTS

I would like to highlight some papers:

[2] https://www.connectedpapers.com/
[3] https://hotcarbon.org/



Bar chart showing the differently labelled entries of my literature study

Figure 3.1: Results of the literature study

GreenSlot: Scheduling energy consumption in green datacenters[0] According to the analyzed papers, this seems to be the first paper that deals with carbon aware scheduling by implementing it as a *Slurm* plugin (See section 4.2.1 for further information on Slurm and such plugins) In contrast to our scenario, where we try to optimize carbon emissions via the public electricity grid, GreenSlot is about datacenters having their own renewable energy production (solar panels on the roof). Using weather data, *GreenSlot* would then predict when solar energy production is high, scheduling jobs to those time frames.

The War of the Efficiencies: the Tension between Carbon and Energy Optimization [0] outline the different ways of carbon aware computing. Among those are *temporal shifting*, the idea that jobs can be executed later when energy is more carbon efficient, is also the main idea for my work. They also use *spatial shifting*, moving jobs across the globe to areas where higher carbon efficiency is possible. *Resource scaling* uses dynamic amounts of hardware according to carbon emissions. In the end, *rate shifting* is the idea to also scale hardware frequencies. During carbon-efficient times, CPU speeds would be increased, leading to faster processing speeds and more energy usage.

All of these techniques are then tested under various parameters. My work will only make use of the temporal shifting, and abstract away the other methods of further saving carbon.

Let's wait awhile: how temporal workload shifting can reduce carbon emissions in the cloud This paper will be built upon in my work the most. [**wiesner_lets_2021**] uses a simulation approach, to simulate temporal shifting. Their workload model consists of known length jobs that can use *checkpoint & restore* to be executed at different time slices. They would further use different traces and test these traces under the assumption of different job-deadlines, meaning that each job would have to be completed by a certain timeframe, and also different regions of the world as described in the background part. Their main takeaways are that increased deadlines lead to reduced carbon emissions, but that this effect also has diminishing returns. They also deduced that regions such as California, with high amounts of solar power, have higher potential for carbon-savings in comparison to nuclear-heavy regions such as France.

Think about linking this to the earlier part

4 Methodology

Based on our related work, describe (using the goals outlined in the beginning) the approach I take: creating a better model to be used in carbon-aware scheduling using the model and evaluating it.

4.1 Improving the current Job model

We should probably argue why the current job model used in literature is not sufficient (WaitAWhile[**wiesner_lets_2021**] assumed constant power usage and no overhead from stopping and resuming)

4.1.1 Power Measurements on Machine Learning Jobs

Options for measuring power There are multiple options for measuring the power of a given computer. One way of classifying these options would be under them being either *logical measurements* or *physical measurements*.

Logical ones would create a model on some metrics and derive the used power. One example would be using Linux' *perf* tool to read hardware performance counters.

Advantages of choosing a logical approach would be that no external hardware is needed and that the overhead of the measurement would be low, as the hardware counters are being kept track of anyway. Disadvantages on the other hand would be that such a model would have to be created or chosen and would include some form of error as all models do.

Physical measurements follow another route; measurement devices would be put between the operating hardware and the power supply. The point where a power measuring device is inserted would dictate what could and could not be measured, a wall mounted measurement device could only measure all power going into a computer and not differentiate between individual programs.

Advantages of physical measurements are that they can give a more holistic measurement of a system as would be the case for a wall mounted measurement device. Portability is an issue however, unlike operating-system supported tools such as perf, a measurement device would need more effort to be used on another system (or be entirely not useable, for example when such devices are only rated for a certain power level).

Due to having a power-measurement tool on-site in our university and it allowing whole-system measurements directly, I chose to follow the physical measurement option.

Measurement tool The concrete tool used is the *Microchip MCP39F511N Power Monitor (henceforth called MCP)*, which can be inserted between the device to test and the wall

Für die Durchführung von Messungen ist es zwingend erforderlich, die verwendete Testumgebung (Hardware wie auch Software) zu dokumentieren sowie Messparameter gewissenhaft zu wählen. Zu den wichtigsten Parametern gehren die Anzahl der Messwiederholungen, die Anwendung von Warmup-Läufen sowie die Identifikation und Vermeidung möglicher störender Einflüsse.

I feel like I should add a paragraph somewhere explaining what I want to measure

This needs some more here

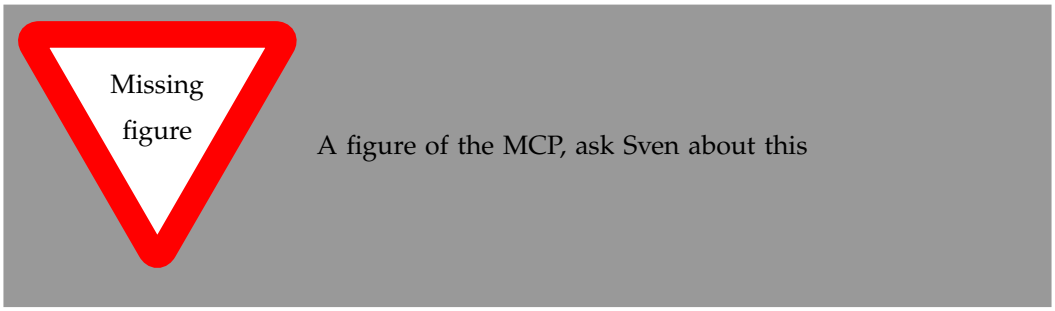


Figure 4.1: The MCP

mounted power supply. A picture of it can be found in figure 4.1. The MCP can report the current power consumption in 10 mW steps, each 5ms.

I then used *pinpoint*, a tool for energy profiling that can use different inputs, among them being the MCP, to read out its data.

I could include an extra paragraph on why the MCP is cool, and what it does differently, perhaps. Was there anything cool? I vaguely remember some measurement devices having two capacitors to more accurately determine usage

The test environment The experiments were run on my personal computer, the components of that are listed via the *hwlist* tool, with unnecessary columns and rows being redacted for brevity:

Listing 4.1: Hardware that was measured

```
   $ lshw -short -C processor -C memory -C display -C bus
Class          Description
==========================
bus            AB350 Gaming K4
memory         16GiB System Memory
processor      AMD Ryzen 5 1600X Six-Core Processor
display        GP104 [GeForce GTX 1070]
```

Information about the operating system is given via *hostnamectl*, again some parts redacted:

Listing 4.2: Used operating system information

```
   $ hostnamectl
  Operating System: Ubuntu 24.04 LTS
            Kernel: Linux 6.8.0-39-generic
```

Measured Program Machine learning (ML) was used as the main motivation for checkpoint & resume scheduling in the related works[**wiesner_lets_2021**] and thus was also chosen by me to be measured and modeled.

The concrete model and framework is secondary for our measurement. In my case, a small model would be chosen in order to have fewer data points for processing as well as faster iterations on the measurement script.

There is a vast amount of machine learning frameworks. For a high-level model, the feature set of the framework only needed to support checkpointing, resuming, and some basic form of logging. Glancing at the documentation of popular frameworks such as *torch*, *tensorflow*, and *huggingface* shows that these features are commonly supported.

With not much bias towards any framework, huggingface was chosen because my supervisor Felix supplied a sample "hello-world"-esque machine learning script for python *roberta.py*[4]

The huggingface trainer supports callbacks, I thus modified the code by adding timestamped logs. These "Events" would be output into another .csv File I could later use.

Later use for WHAT?!

Conducted Experiments A script[5] was created to execute each experiment.

On a high-level view, the following experiments were conducted:

1. Run the whole program start to finish
2. Run it partially, checkpointing after some step, sleeping, resuming from that step
3. Run it partially, checkpointing after some step but aborting before the next checkpoint. Then resume as above.
4. Run only the startup phase up until the ML would start
5. Do nothing, measure the system at rest

Experiment 1 would give a baseline for what the job would look like without checkpoint & resume. Number 2 and 3 would be used to determine the overhead of checkpointing the job. 4 would be used to validate the other ones. The last experiment is necessary to determine the baseline energy consumption of the environment.

To execute these experiments inside a repeatable bash script, additional command line parameters were added to the given python script. For example, there would be a boolean parameter `--resume_from_checkpoint`, or an integer parameter `--stop_after_epoch` would be used for experiment 1 to 2. The way of doing experiment 4 was to copy the script, and delete everything after the imports.

Creating repeatable measurements As this is being run on standard hardware on a standard operating system, all experiment are subject to noise. For example, *Dynamic frequency and voltage scaling (DFVS)*, the OS technique of increasing CPU "speeds" according to work load would add power in an uncontrolled way. Also, background tasks may happen "randomly", increasing power usage.

Thus, for the testing, any foreground apps would be closed. I also used *cpupower*, as shown in snippet below, to set the CPU frequency to a set value:

Think about whether this is really interesting, I guess keep it if I need more content

Listing 4.3: 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' \
```

What is the MAXFREQ of my ryzen 5?

[4] https://github.com/Quacck/master-thesis/blob/main/power-measurements/roberta.py
[5] https://github.com/Quacck/master-thesis/blob/main/power-measurements/measure_roberta.sh

```
        | 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
```

As machine learning makes use of available GPUs, the frequency should also be similarly set to a defined value. NVIDIA provides guide on how to do so.[6] Sadly, my used GPU, the NVIDIA GTX 1070, is not capable of fixing the frequency as of the time of conducting these experiments. While it is supposed to be possible according to SOURCE, but there seems to currently be drivers issue preventing this. Thus, the frequency of the GPU was not fixed. To reduce the effect of frequency scaling here, the time between experiments was increased generously so that any impact from such scaling would reoccur throughout each run and there would be reduced dependency between runs.

Reconstruct this argument, perhaps theres still something in my PC history | grep

Find the forum post of people complaining

further explain how the experiments were conducted

Conducting each experiment Each experiment was re-run 10 times. Between each run, there would be a `sleep` period of 10 seconds and one of two minutes in the partial executions. Additionally, `pinpoint`'s feature of measuring before and after the actual program-to-test would be used. This leads to a period of 30 seconds being measured around the actual experiment. Plotting these additional time frames would give a quick visual indicator whether experiments are sufficiently isolated from each other, ergo when the power draw is at the baseline as the actual program starts.

As there is some data being downloaded and persisted during the execution of the ML, before each run, the data would be cleaned up.

Collected data For each experiment, a named and timestamped folder would be created in the `/power-measurements` folder of my repository. Each folder would then hold a `.csv` with pinpoint's timestamped power measurements. The added timestamped-logging would be saved into another `.csv`. While figuring out how to conduct the experiments, I would then plot these measurements early and visually spot if there were any obvious errors or mistakes.

Determining the baseline power draw Beginning with the most exiting experiment, determining the baseline and testing the amount of the underlying environment. One sample run is shown in Figure 4.2. The blue dots in figure represent each data point. The red line is a smoothed Gaussian trend line with $\sigma = 2$. Dark-green vertical lines are the logged or derived "events" for each run. In this case, nothing happens, so it is only the start and end of `sleep 120`. Notice how the trace starts 30 seconds before the start and continues for another 30 seconds because of the aforementioned `pinpoint` feature. Going further, these

[6] `https://developer.nvidia.com/blog/advanced-api-performance-setstablepowerstate`

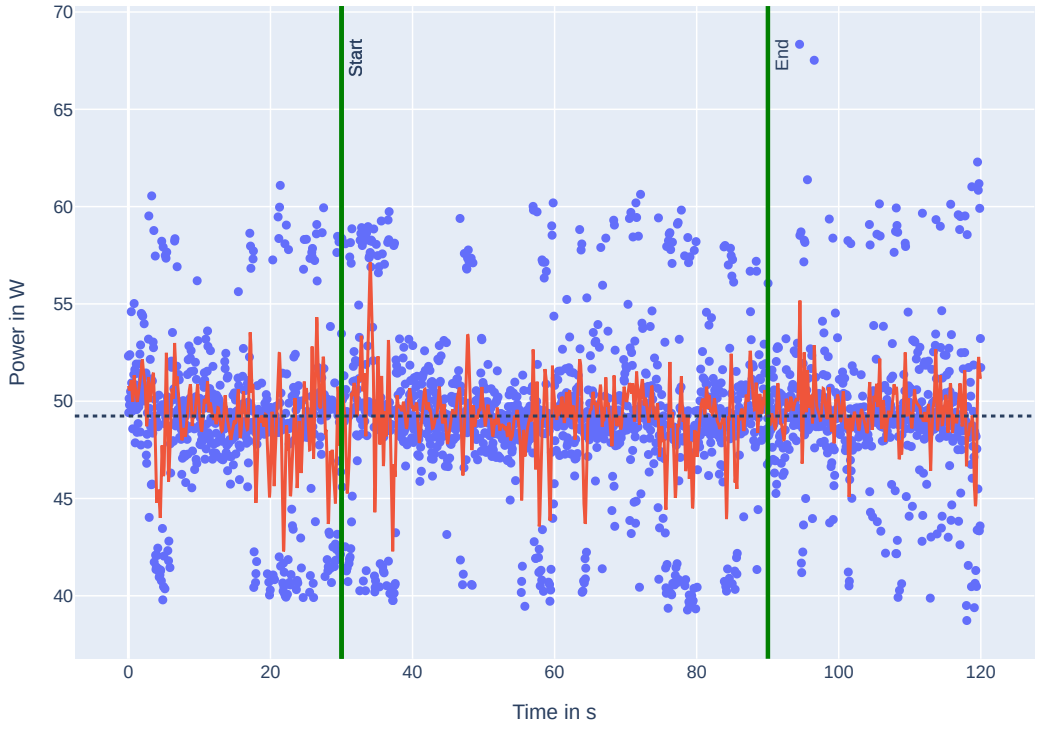


Figure 4.2: Sample run of the baseline experiment

additional measurements will be redacted for brevity, unless something worth mentioning happens outside the actual experiments.

Check later if something interesting happens or if I should reword this.

Across all 10 runs, the average baseline power draw is calculated via the mean of all data points. This comes out as an average of 49.8 W with a standard deviation of 4.4 W.

The baseline power draw will be less interesting going further, but will put perspective on the power draws of the other experiments. The standard deviation should give a broad idea of how much noise is in the system environment.

The non-interrupted run For the non-interrupted machine learning run, a figure of a sample run is provided in 4.3. Figure 4.4 shows the stacked trend lines of the 10 different runs. For simplicity's sake, I refrained from doing more elaborate statistical analysis on the different runs as the visual check of them being very similar seemed enough. There was an option to discuss the measurement results after normalizing each measurement point to its phase, but that would not change the result.

What?

The main takeaways from these measurements are:

1. There is a long (about 25%) start-up phase, which is spent in starting python, loading libraries, and loading data to disc.
2. There are periodical work-phases; a high-power training phase would be followed by low-power evaluation phase and a low-power checkpointing.

Uhm, where did my evaluation phase markers go?? They are mushed together as they share a label

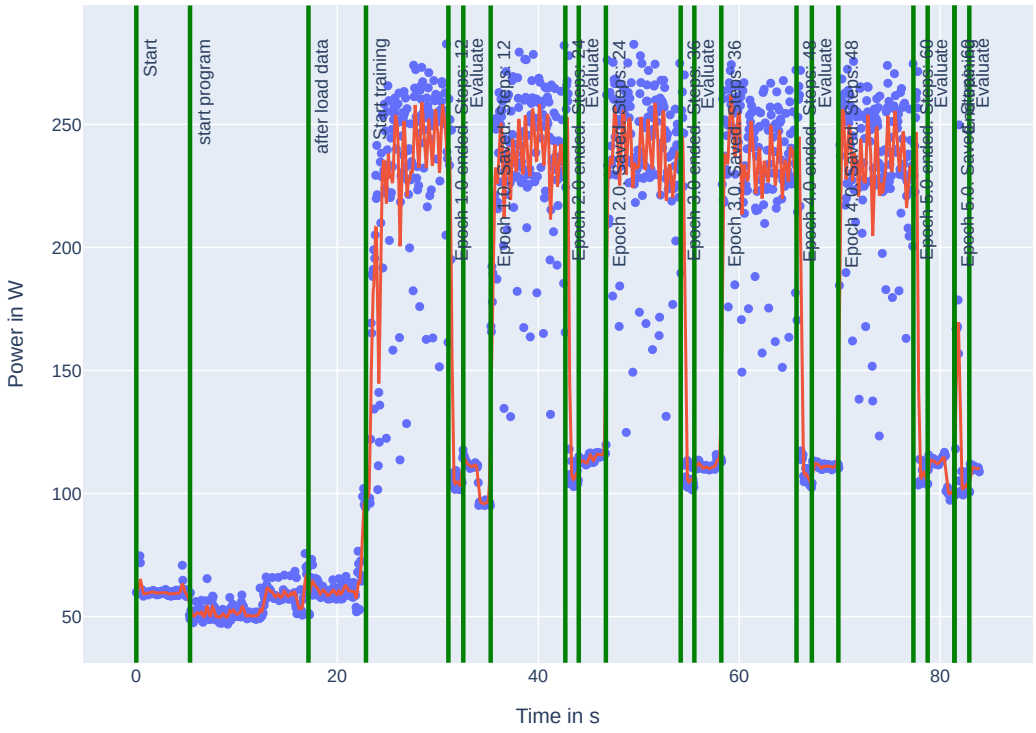


Figure 4.3: Sample run of the full run experiment

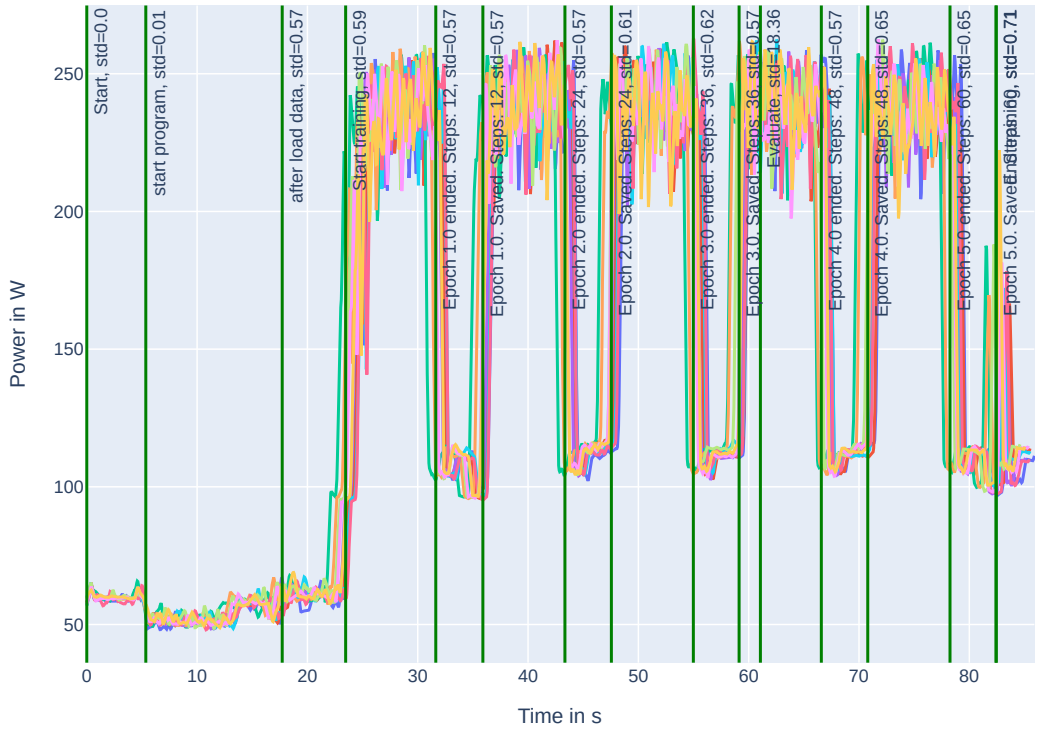


Figure 4.4: Stacked trend lines of the power consumption for the full runs

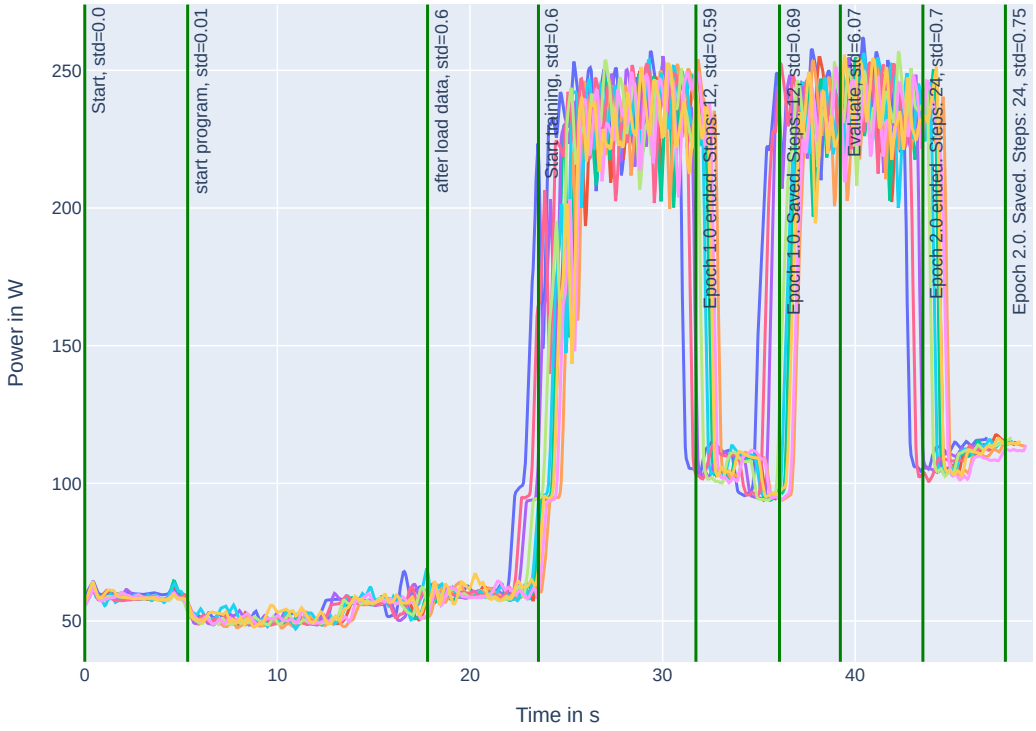


Figure 4.5: Power draws of the ML up until stopping after epoch 2

3. There also appears to be higher variance in measurements during the training phases in comparison to the others.

This already shows, that improvements upon the constant-power model used in [**wiesner_lets_2021**] are possible. For example, in this case the start-up phase has a much lower power-draw than the work-phase.

Results of the checkpoint and restore experiment Similarly to before, results will be discussed using the stacked plots 4.5 and 4.6; each individual run is plotted in the repository, however.

Here we can observe that

1. The amount of work done is the same. Similarly to the full-run experiment, the ML still takes the full 5 epochs and has the same work-phases
2. There is no overhead from checkpointing itself, as the checkpoints are being created regardless of them being resumed from later.
3. Resuming the jobs results in an added start-up phase. This phase is slightly shorter by a few seconds than the ones in the full runs, likely due to not needing to download the dataset again.

Additional considerations are that the amount of epoch that are worked off before checkpointing and resuming may matter, but it will be assumed that this is not the case.

I could just run this again with another epoch parameter

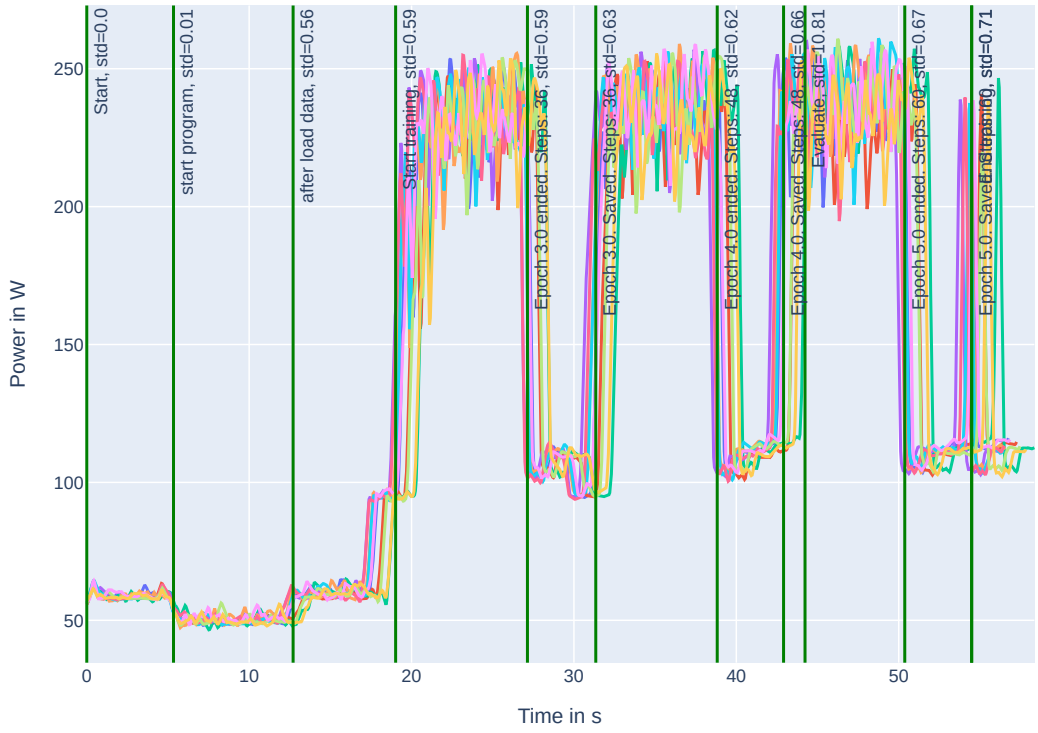


Figure 4.6: Power draws after continuing from the second checkpoint

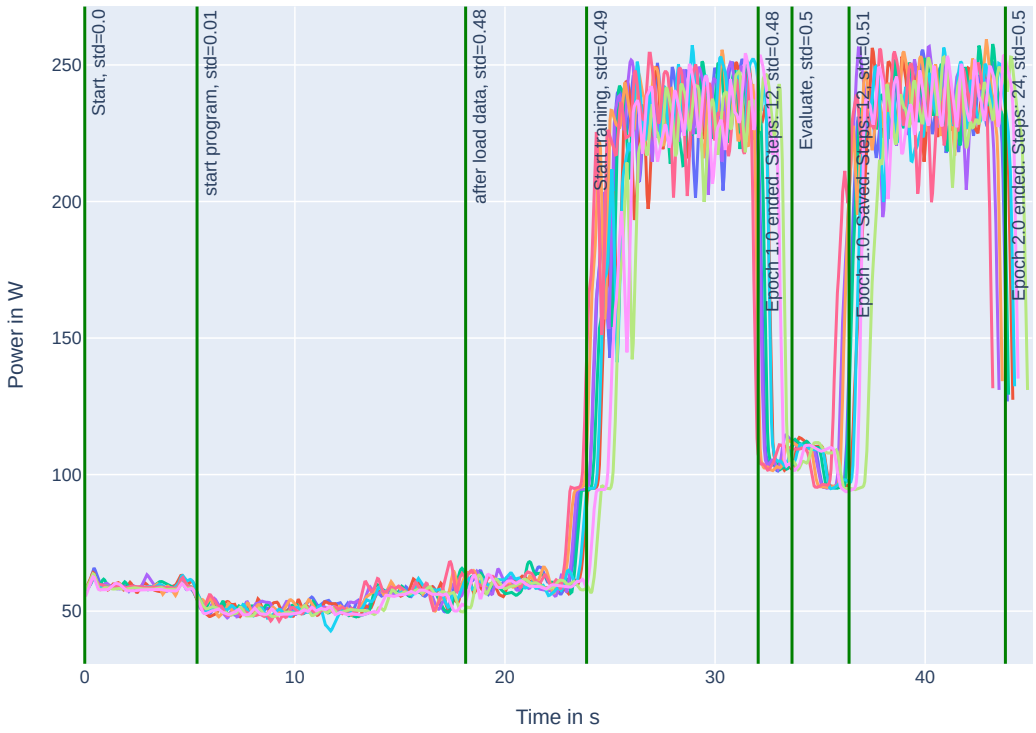


Figure 4.7: Power draws of the ML up until stopping after epoch 2

Results from checkpoint and resume with abort Unlike the previous experiment, where work is stopped as soon as checkpoint is created, this time the program will be stopped just before a checkpoint is created (in this case just the second checkpoint would be saved). This should show the maximum created overhead from a checkpoint & resume strategy.

While this may sound artificial, it could happen in environments where the interaction between the scheduler and the job is not well orchestrated, for example in an environment where jobs are stopped "at random" like in a cloud spot instance..

Improve the wording here

Again, the results are visualized in figure 4.7 and 4.8. Attention should be paid to:

1. The behavior of the repeated start-up phase is kept
2. There is now a full additional training phase added to the overall work

Calculating the energy costs of each run To support the argument that there is indeed an overhead from checkpointing & resuming, I calculated the energy needed via integral of each experiment:

- unstopped costs 12.97 kJ on average with an std of 0.04
- save+resume costs 13.94 kJ on average with an std of 0.1
- save+abort+resume costs 15.72 kJ on average with an std of 0.07

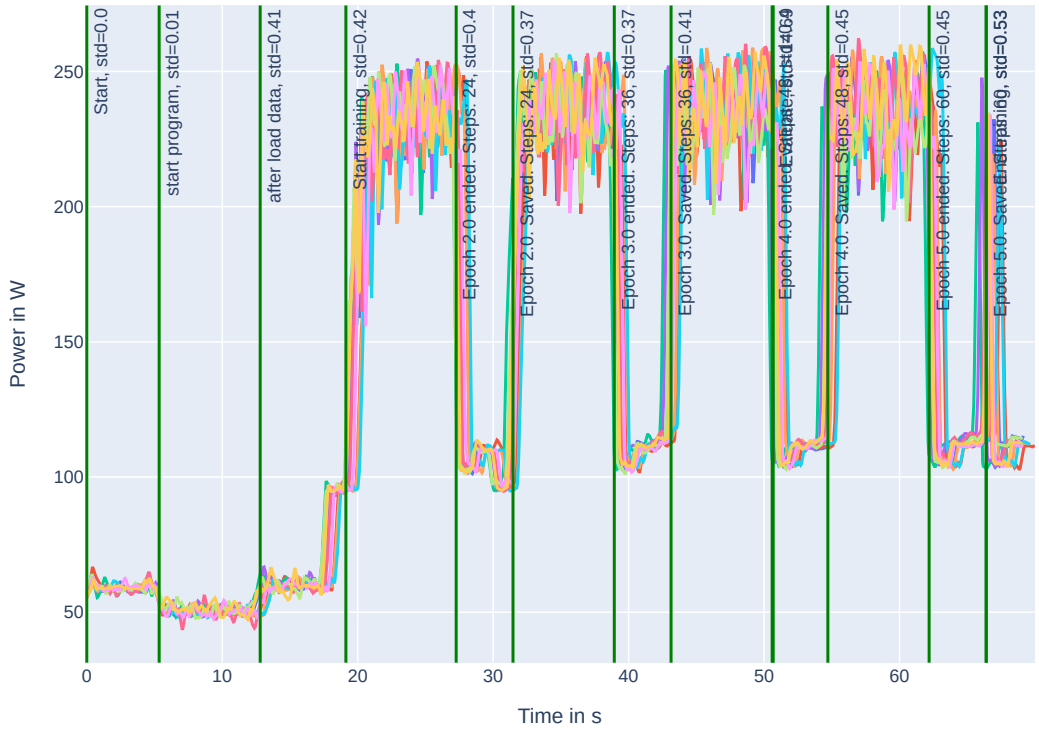


Figure 4.8: Power draws after continuing from the second checkpoint

4.1.2 Defining a new model

Now that we know what a high-level job looks like, we can pick it apart and reduce the real-world measurements of one program to a more generic model.

Summarizing the findings from the previous paragraphs; it was shown that

- The given job has phases that have different power draws
- Checkpointing & resuming carries overhead in the form of startup costs and possible wasted work.

The parameters for the improved job model are shown in form of the python implementation in 4.4.

Check whether this ref works

Probably need to improve the code here

Listing 4.4: Python Model definition

```
class Phase(TypedDict):
    name: str
    duration: float
    power: float
    is_checkpoint: NotRequired[bool]

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

Some simplifications are made: the duration of each phase is well known and the power per phase is a constant. Phases can also be named for later reference. Using these specifications, a simple time-to-power function can be defined, that looks up the input time and traverses the phases in order. While not very efficient, this works well enough in a simulation environment.

The measurements that have been taken can now be fit into this new model. As each measurement-point can be associated to a phase via the aforementioned added logging, the average of each phase-associated measurement is used to determine the model parameters. The durations of the phases are calculated simularly by taking the average time the logging occured during the measurements.

Using this strategy on the 10 complete runs results in figure 4.9, which shows the derived model on black with the previous figure 4.3 in less opacity. The startup phase looks well approximated, visually however there is some error during the work phases. The training phases each have a high variance, which is not captured by the constant power approximation. After each training phase, the power goes down seemingly liniarly, which is also approxiimated by the constant.

Should probably add the phase markers back in

Model error analysis To analyse the error of this model, I cross validated the power's and total energy's RMSE using `scikit-learn`'s `LeaveOneOut` strategy. The first one would give a measure of the model's accuracy on a short-time (sub-second) scale, the latter would tell the long-time (whole job) scale accuracy of the model.

Each of the 10 runs would be taken as the ground truth while the other 9 would be used to create the model. The results are the following: t power between the prediction

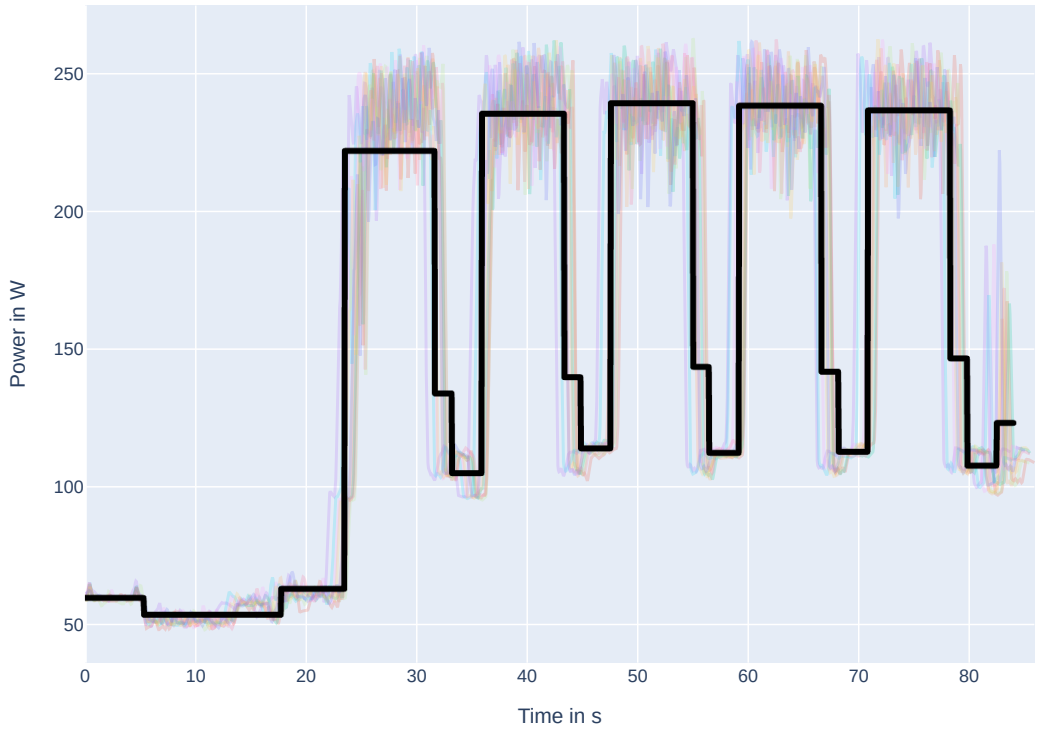


Figure 4.9: Model of roberta.py (black) vs. all measurements

and remainder has an RSME of 39.3 W while the difference in total energy is calculated as -0.1 kJ.

Interpreting this, it seems that the model performs poorly as a predictor for die exact power used at some timepoint as the RMSE is rather large (think of the maximum power drawn being about 250W). However, in the context of carbon aware scheduling, this should not be too big of an issue as the time frames for carbon-emissions are order of magnitude larger. The high error likely comes from the high variance during the training phases which is not captured in the model.

The total energy predicted by the model is very close to the actual real life experiment, this should mean that the total carbon calculated on the model should also be close to the carbon emitted by the real program.

- we can deduce phases
- each phase has a constant power draw
- give an example of how to represent the real-world measurements into a model
- now we should proof that the model actually represents the reality to a certain degree (error analysis)
- have a cute graph showing the measurements and the model-"measurements" next to each other
- also show that the stop-resuming functionality can be represented with our model

4.2 Choosing an implementation approach

Now that an improvement on the job model has been made, the question, on how to evaluate the implications of said model, remains.

4.2.1 Carbon-aware scheduling via a slurm plugin

My first idea was to use a non-simulation approach. The HPI's Data Lab[7] runs a *slurm* cluster and also has some nodes with power measurement infrastucture included. Slurm is an "open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters".[8] The job scheduling part is important here, it also supports a plugin infrastructure that includes scheduling plugins. One of the highlighted papers[0] in the related works section specifically used slurm for its carbon-aware scheduler implementation and thus seemed like a good starting point for my own work.

Installing Slurm locally For my purposes, a local installation would suffice as I would not need to run heavy workloads but instead just the scheduling part of slurm. While there is a slurm `apt get` package for my ubuntu version, this would not work as any plugins to be included during the slurm compilation, meaning I would have to do the same.

[7] `https://hpi.de/forschung/infrastruktur/hpi-data-engineering-lab.html`
[8] `https://slurm.schedmd.com/overview.html`

Slurm's documentation provides some guidelines on how to install slurm, which I followed. I gave cloning the main branch an attempt but I would get stuck during the compilation process, using the predefined released versions however worked.

Installing `munge`[9] is also necessary, which is used for authentication in slurm.

One problem arose as I tried to start the `slurmd`- and `slurmctld` services. The first one is the worker service that would later execute jobs submitted to slurm. The latter is the main controller that, for example, schedules jobs on workers. While the command to start them would not fail, upon node inspection via Slurm's `scontrol` command, it would show that all nodes are `DOWN` instantly. | Dealing with slurm's problems usually leads to inspecting its logs, in my case the logs showed the following:

Listing 4.5: Slurm error logs

```
$ less config.log

error: Couldn't find the specified plugin name for cgroup/v2
    looking at all files
error: cannot find cgroup plugin for cgroup/v2
error: cannot create cgroup context for cgroup/v2
error: Unable to initialize cgroup plugin
error: slurmd initialization failed
```

Slurm uses Linux' `cgroup` feature to manage the submitted job's hardware ressources. The log hints at some problem related to slurm's usage of it. The solution was this was to provide slurm's `cgroup.conf` file. In my use-case of getting slurm to simply start, this did not need to be very sophisticated so I just used an off-the-shelf (off-the-stackoverflow[10]) configuration file.

Running `scontrol` again, I was finally able to see idling nodes, meaning that slurm was succesfully installed form source:

Listing 4.6: Slurm running

```
$ scontrol
scontrol: update NodeName=vincent-Laptop STATE=RESUME
```

Creating a scheduler plugin The slurm documentation provides a short guide on how to add a plugin to slurm..[11] As a start, I simply copied slurm's default scheduler (which is also a plugin) to to the specified directory under a new name, and addings that new name to slurms build files. It was then was time to recompile slurm. Now however, during the recommended `autoreconf` step, an error occured:

Listing 4.7: Plugin recompilation errors

[9] `https://github.com/dun/munge/wiki/Installation-Guide`
[10] `https://stackoverflow.com/a/74211989`
[11] `https://slurm.schedmd.com/add.html`

```
$ autoreconf
auxdir/x_ac_sview.m4:35: warning: macro 'AM_PATH_GLIB_2_0'
    not found in library
configure:25140: error: possibly undefined macro: AM_PATH_GLIB_2_0
      If this token and others are legitimate,
      please use m4_pattern_allow.
      See the Autoconf documentation.
autoreconf: error: /usr/bin/autoconf failed with exit status: 1
```

The solution, while not very obvious, was to install the `libgtk2.0-dev` library.:[12] I then added a simple logging which then showed up in slurm's log files too.

Adding more logic to the scheduler plugin One very helpful step for developing inside slurm is to enable the debugging flags. This must be decided before compilation by using the `--enable-developer` and `--disable-optimizations` flags during the `/configure` step. With that, debug symbols are added to the outgoing binaries. As I was using vscode, I could then attach its debugger to the running slurm thread with full functionality.

The code of the plugin runs in its own thread and there is no sandboxing or similar around it. Thus there are seemingly no limitation on what can be done inside the plugin. For testing, I read out information on the incoming jobs such as set constraints or the user supplied comments. Terminating the jobs was also possible inside the plugin.

Problems of a scheduler plugin One big problem manifested in that not all jobs "showed up" inside the plugin's job queue. If I would submit 6 jobs, via slurm's `squeue` command, only a part such as the last 3 could be logged inside the plugin. A possible explaination for this could be slurms scheduler architecture: while there is a scheduler plugin, there also is a scheduler inside slurm's main loop. That main scheduler loop also uses the same job queue as the plugin.

To hack around this, I tried disabling slurm's main scheduling loop by setting `sched_interval=-1` inside the slurm configuration file. While this had the effect of being able to access all incoming jobs inside the plugin, it also had the side-effect of disabling all logic concerning starting the jobs. So by choosing this route, the plugin would apparently also need to re-implement alot of extra logic, which conventionally would be not be put inside the scheduler plugin.

I also looked into whether there were any API hooks that are exposed to the plugin. Up until slurm version 20.11, scheduler plugins had callbacks such as when jobs were submitted. There also was support for "passive" scheduler that would get invoked when determined by slurm. The version I tested however, 23.11, removed all such functionality and documentation. All plugins are implemented via threads that only have callbacks on when they are started and stopped.

Thus, since there was no apparent way of getting around this scheduler race condition between the plugin and the main loop. The scheduler approach as a whole was dropped. While I would not say that a plugin approach is impossible, the effort to implement one from scratch subjectivly seems very high. The public documentation for developing on

[12] `https://stackoverflow.com/questions/7805815/autoconf-error-on-ubuntu-11-04`

slurm is scarce. There is a mailing list that can be searched, but it looks to be mostly aimed for administrating slurm and not developing it.

Other avenues that could be explored are slurm's Lua plugins. There is also a *slurm simulator*[13] which could potentially be used for carbon-aware scheduling simulation, but that I did not look into much for reasons of little documentation and seeming lack of continued support.

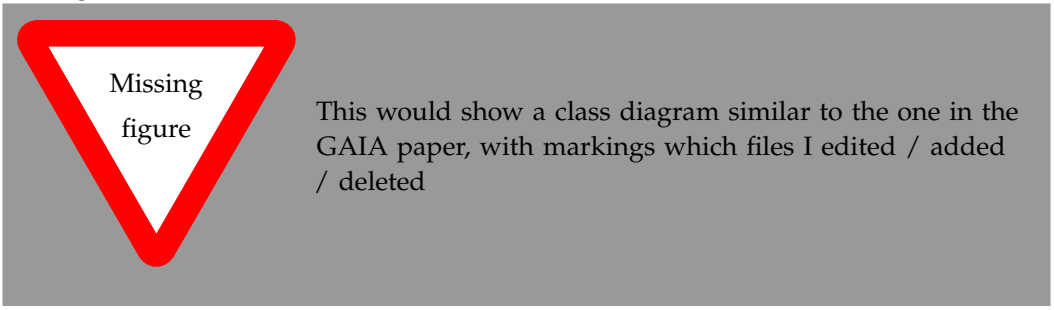
4.2.2 Using a Simulation approach

Thankfully, just at that time, a new paper [0] was released. They made a prototype testbed for simulating job scheduling on cloud providers. These Jobs could be executed on spot instances (cheap VMs that seek to increase cloud utilization), on-demand instances (short-notices VMs that are thus more expensive) or pre bought VMs (medium cost, but may be wasted), the paper then discussed balancing carbon- and dollar costs. They included a small notion notion of hardwre requirements in the form of required CPUs per job, but that was only used to scale the cost; all hardware requirements were abstracted away in the form of the cloud always having computing ressources available.

The important part is that they also included an implementation of some schedulers used in the highlighted related works. Among that an implementation for WaitAWhile[**wiesner_lets_2021**]. This meant that I could add the improved job model to that exisiting testbed and have something to compare against aswell.

Description of the GAIA simulator I would first like to describe the existing testbed in detail to make it clear which part is my work and which is not.

The way to do this section would be to a) describe what was already there, and b) what I changed



Describe the figure, explain why I am for example removing the part about the dollar-costs and the slurm-scheduler adapter. I should also describe which parts of the program I am modifiying to tackle my Forschungsfragen

Stuff I should describe about the simulation before I added anything:

Assumptions of the simulation

- Joblängen sind bekannt
- Jobs können zeitlich verschoben werden (begründet daraus, dass sie als Batch Jobs submitted werden, andere Jobs werden hierbei nicht betrachtet)

[13] https://hpckp.org/articles/how-to-use-the-slurm-simulator-as-a-development-and-testing-environment/

- User geben dabei an, wie lange der Job verschoben werden darf
- Die carbon curve auf dem electrical grid ist für kurze Zeiträume in der Zukunft bekannt
- Die Hardware ist zZ nicht begrenzt. Das war in der related work auch nicht so. Eigentlich wäre es spannend sich das anzuschauen, allerdings sind die bisherigen Scheduler halt darauf garnicht gemünzt, da werden alle Jobs unabh. voneinander gescheduled. Man könnte das via publicCloud argumentieren, allerdings wäre das questionable, in wie fern der scorelab trace benutzt werden kann (da das ja auf in einem lokalem datacenter läuft)
- TODO: Joblängen sollten dem Scheduler nicht bekannt sein. Die Workloads aus GAIA werden allerdings so gescheduled als ob man perfekte Knowledge hat. Das reicht zwar für ein upper bound an carbon savings, ist aber nicht sehr realistisch.

Data being used here i could describe which data is already being used (the traces, aswell as the historical carbon data)

- Welche Traces gibt es, wodurch werden die characterisiert? (Länge, Anzahl, etc, etc) Vllt. kann man hier nen coolen vergleich erstellen, Auch könnte man ein paar Sätze darüber schreiben, wie die bisher in GAIA aufgenommen wurden.
- Wie den scorelab trace benutzen und übersetzen? Gerne auf ner halben Seite aufschlüsseln, was die einzelnen Attribute aus sacct bedeuten.
- Ansonsten kann man noch die dynamic ernergy sachen als Datenquelle auflisten, bzw. das mini experiment mit fmnist und roberta

4.3 building ontop of the existing gaia sim

Which parts of GAIA do I add on? => this should just be the schedulers and the part where the carbon is calculated, this ensures that

4.4 Evaluating carbon-aware scheduling with the new job model

Hi!

5 Results

- here we would try to show off the difference between power-and-phase-oblivious scheduling and my new implementation which can make use of that
-

6 Discussion

welche schlüsse können wir aus den ergebnissen ziehen?

7 Future Work

lol

Bibliography

[0] Walid A. Hanafy, Roozbeh Bostandoost, Noman Bashir, David Irwin, Mohammad Hajiesmaili, and Prashant Shenoy. "The War of the Efficiencies: Understanding the Tension between Carbon and Energy Optimization". en. In: *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. Boston MA USA: ACM, July 2023, pages 1–7. ISBN: 9798400702426. DOI: 10.1145/3604930.3605709 (cited on page 10).

[0] Walid A. Hanafy, Qianlin Liang, Noman Bashir, Abel Souza, David Irwin, and Prashant Shenoy. "Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. Volume 3. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pages 479–496. ISBN: 9798400703867. DOI: 10.1145/3620666.3651374 (cited on page 27).

[1] Andrew S. Tanenbaum and Albert Woodhull. *Operating systems: design and implementation: [the MINIX book]*. en. 3. ed. The MINIX book. Upper Saddle River, NJ: Pearson Prentice Hall, 2006. ISBN: 978-0-13-142938-3 978-0-13-505376-8 978-0-13-142987-1 (cited on page 7).

[0] Íñigo Goiri, Kien Le, Md. E. Haque, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. "GreenSlot: scheduling energy consumption in green datacenters". en. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. Seattle Washington: ACM, Nov. 2011, pages 1–11. ISBN: 978-1-4503-0771-0. DOI: 10.1145/2063384.2063411 (cited on pages 10, 24).

Eidesstattliche Erklärung

Hiermit versichere ich, dass meine Arbeit zur Erlangung des Grades “Master of Science” der Digital-Engineering-Fakultät der Universität Potsdam mit dem Titel “A parameterized test bed for carbon aware job scheduling” (“Ein parametrisierbares Testbed für kohlenstoffbewusste Jobplanung”) selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, den 9. September 2024

_____

(Vincent Opitz)