FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lehrstuhl für Informatik 4 · Systemsoftware

Michael Gebhard

Carbon-Aware Systemd Timers

Masterarbeit im Fach Informatik

3. November 2024

Please cite as:
Michael Gebhard, “Carbon-Aware Systemd Timers”, Master’s Thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Dept. of Computer Science, November 2024.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik
Systemsoftware
Martensstr. 1 · 91058 Erlangen · Germany

Carbon-Aware Systemd Timers

Masterarbeit im Fach Informatik

vorgelegt von

Michael Gebhard

angefertigt am

Lehrstuhl für Informatik 4
Systemsoftware

Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer:	**Luis Gerhorst, M.Sc.**
Betreuender Hochschullehrer:	**Prof. Dr.-Ing. Rüdiger Kapitza**
Beginn der Arbeit:	**4. Mai 2024**
Abgabe der Arbeit:	**4. November 2024**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties. I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Michael Gebhard)
Erlangen,
3. November 2024

ABSTRACT

Carbon emissions caused by electricity generation contribute significantly to anthropogenic climate change. Electricity-grid carbon intensity varies over time and between different regions. Recent studies exploit variations in carbon intensity to schedule datacenter workloads towards locations and times of low carbon intensity, thereby reducing operational carbon emissions incurred by these workloads. While these approaches cover datacenter use cases, total electricity consumption of datacenters was estimated at 130 TWh in 2018. In comparison, electricity consumption of consumer desktops in 2013 was estimated at 132 TWh, showing that consumption by consumer desktops is at least comparable to, if not exceeding, the consumption of datacenters. I propose CARBON-AWARE SYSTEMD TIMERS to introduce carbon-aware workload scheduling for single-machine scenarios employing the system and service management software Systemd. CARBON-AWARE SYSTEMD TIMERS utilize online grid-carbon-intensity forecasts to reduce carbon emissions by delaying suitable workloads, such as backup, update, file-system or database-maintenance tasks. In a benchmark of a realistic daily backup task, simulated across the full year 2023, emission reductions of 30% were achieved in comparison to the default configuration of this daily backup service. CARBON-AWARE SYSTEMD TIMERS require no region-specific configuration of individual workloads. Minimal setup is required by system administrators or package maintainers to mark workloads as suitable for carbon-aware delays. CARBON-AWARE SYSTEMD TIMERS deliver the benefits of carbon-aware workload scheduling to single-machine scenarios, while imposing minimal setup overhead for machine administrators.

KURZFASSUNG

Treibhausgasemissionen verursacht durch Stromerzeugung sind ein bedeutender Faktor für anthropogenen Klimawandel. Die spezifische Carbon-Emission der Netzstromerzeugung (CO_2-Intensität) schwankt über Zeit und zwischen unterschiedlichen Regionen. Aktuelle Forschungen nutzen diese Schwankungen, um Rechenaufgaben in Rechenzentren zu Orten und Zeiten mit geringer CO_2-Intensität zu verschieben. Diese Ansätze decken zwar Rechenzentren ab, dabei wurde der Gesamtstromverbrauch von Rechenzentren 2018 auf 130 TWh geschätzt. Im Vergleich wurde der Stromverbrauch privater Rechner 2013 auf 132 TWh geschätzt, was den Stromverbrauch privater Rechner mindestens vergleichbar zu dem Stromverbrauch von Rechenzentren stellt. Mit CARBON-AWARE SYSTEMD TIMERS präsentiere ich CO_2-bewusste Ablaufplanung für Einzelrechner, welche die Systemverwaltungssoftware Systemd verwenden. CARBON-AWARE SYSTEMD TIMERS nutzen online CO_2-Intensitätsvorhersagen zur Minderung von Treibhausgasemissionen durch Aufschieben verzögerbarer Rechenaufgaben, wie zum Beispiel Datensicherungs-, Softwareaktualisierungs- und Datenwartungsaufgaben. In der Simulation einer realistischen täglichen Datensicherungsaufgabe über das Jahr 2023 hinweg, wurde eine Minderung der CO_2-Emissionen um 30% erreicht, verglichen mit den Werkseinstellungen dieses Datensicherungsdienstes. CARBON-AWARE SYSTEMD TIMERS erforden keine regionsabhängige Einstellung einzelner Rechenaufgaben. Lediglich minimaler Einrichtungsaufwand, durch Systemadministratoren oder Paketinstandhalter, ist notwendig, um verzögerbare Rechenaufgaben als solche zu markieren. CARBON-AWARE SYSTEMD TIMERS bringen die Errungenschaften CO_2-bewusster Ablaufplanung zu einzelnen Rechner und erfordern dafür nur minimalen Einrichtungsaufwand der Verwalter dieser Maschinen.

CONTENTS

1 INTRODUCTION

Greenhouse gas emission tied to electricity generation play a major role in anthropogenic global warming. The computational capacities of datacenters as well as consumer computers are ever-increasing. The associated electricity consumption makes computers and, in turn, the software running on them a non-negligible contributor to climate change. Carbon-aware software exploits knowledge of the regional carbon intensity of grid electricity to reduce carbon emissions cause by execution of a workload while maintaining performance aspects.

Carbon-aware workload scheduling is an emerging technology in large-scale datacenters [Rad+22]. With datacenter electricity usage estimated at 1% of total electricity usage in 2018, carbon awareness can have a significant impact on environmental, economic and electricity grid stability concerns [Wie+21]. In absolute numbers datacenter electricity consumption was estimated at $130TWh$ in 2018 [Mas+20]. In comparison, electricity consumption of consumer desktops was estimated at $132TWh$ in 2013 [CA13]. Given that consumer desktops are comparable to, if not exceeding, datacenter electricity consumption, I propose CARBON-AWARE SYSTEMD TIMERS[1] to bring the benefits of carbon awareness to single-machine scenarios running Systemd, such as a large portion of consumer computers as well as private and corporate owned non-datacenter servers.

1.1 Problem Statement

Carbon awareness in datacenters may facilitate power modeling, geographic [Rad+22] and temporal shifting [Wie+21]. Power modeling allows a datacenter to reduce carbon impact by utilizing slower, more efficient hardware during high grid carbon intensity times and optimize effectivity during low intensity times by means of faster, possibly less efficient hardware. Geographic shifting facilitates relocation of workloads between geographically separate datacenters to utilize regional differences in carbon intensity. Temporal shifting improves the carbon footprint by delaying workloads from high intensity to low intensity times.

Power modeling and geographic shifting are applicable to any workload, while respecting performance constraints. However, neither of the two are applicable to single-machine scenarios such as consumer desktops.

Temporal shifting in contrast can be applied in a single-machine scenario, but is only meaningful if deadlines associated with a workload allow shifting of at least several hours [Wie+21]. Information about such deadlines is readily available in a datacenter context in terms of Service Level Agreements (SLA). In the context of a consumer or single-server corporate setting, deadlines may need to be assumed or configured on a per-use-case basis. Examples of shiftable workloads with typically

[1] `https://gitlab.cs.fau.de/carbon-aware-systemd-timers/carbon-aware-systemd-timers`

flexible deadlines are backups, updates and file system or database maintenance jobs running at a daily or larger interval. In summary, delaying suitable workloads offers opportunities to reduce carbon emissions without impacting functionality.

Management of workloads consumer machine and singular servers is a task of the machine's service management. In most Linux distributions, such as Debian, Ubuntu, Fedora, and Arch Linux, Systemd provides this service-management functionality. Service management includes timers to execute services at specific wall times or intervals, e.g., a backup job at 12 AM every day. Systemd has no concept of deadlines with respect to when such a timer should expire. However, a human operating this system may expect this daily backup job to be executed at least within the same day. Exploiting the flexibility of these temporal constraints allows the necessary delays to achieve reductions in carbon emissions.

This work addresses the opportunities for carbon emission reductions through delaying workload execution, while minimizing the effort for machine administrators and software maintainers to benefit from this solution.

1.2 Approach

In order to optimize the carbon footprint of Systemd services, CARBON-AWARE SYSTEMD TIMERS offer modular interfaces for carbon intensity prediction, resource usage collection and carbon-aware timer management, along with individually replaceable implementations of all three. Carbon intensity predictions from external services, such as Electricity Maps[2] allow scheduling timer expiration into times of low carbon intensity. Collection of service execution durations helps to avoid carbon-aware scheduled services extending into times of high carbon intensity. I combine these two data sources to reduce the carbon footprint of shiftable Systemd services. To estimate the carbon footprint of a service I focus on operational carbon, more specifically electricity consumption of the CPU.

The modularity of these interfaces allows adaptation of this work for different carbon intensity prediction methods. Resource usage collection can be adapted for different optimization goals, such as inclusion of embodied carbon, electricity prices or availability of locally generated electricity.

In addition to flexibility, CARBON-AWARE SYSTEMD TIMERS are transparent to the affected services. This means, workloads can benefit from this carbon footprint optimization, without the need for individual services to be aware of regional aspects of grid carbon intensity. Either maintainers of a service or a machine's administrator only need to declare a service as shiftable by assigning an interval within which the workload may be delayed.

In summary, CARBON-AWARE SYSTEMD TIMERS provide a modular, easily adaptable solution to reduce carbon emissions by temporal shifting of workloads, while requiring minimal adaptation by software maintainers or machine administrators.

[2] Real-time and predictive electricity signals [Ele22].

1.3 Overview

First, I examine the environment in which CARBON-AWARE SYSTEMD TIMERS combine carbon awareness and single-machine system management in Chapter 2. Following up, I present the design starting from a basic use-case and building to the necessary extensions of system management in Chapter 3. In Chapter 4, I dive into the implementation challenges in connecting online carbon intensity prediction and Systemd service management. CARBON-AWARE SYSTEMD TIMERS manage to reduce carbon-emissions of test workloads by $10-36\%$, as I show in Chapter 5. Proceeding in Chapter 6, I examine the differences between CARBON-AWARE SYSTEMD TIMERS for single-machine scenarios and present approaches to carbon awareness in datacenter settings. Finally Chapter 7 concludes this thesis and provides an outlook on further adaptations of CARBON-AWARE SYSTEMD TIMERS.

2 BACKGROUND

In this chapter, I introduce the technical background of CARBON-AWARE SYSTEMD TIMERS. I first present the terminology used to describe the goals of carbon-aware software. Followed by an outline of a subset of Systemd's functionalities, that CARBON-AWARE SYSTEMD TIMERS interact with.

Section 2.1 presents the definitions for carbon awareness, carbon intensity, embodied carbon, and operational carbon of the Green Software Foundation[3]. In particular focusing on operational carbon, as the goal of CARBON-AWARE SYSTEMD TIMERS is to reduce operational carbon emissions of workloads associated with Systemd timers.

In Section 2.2 I outline the structure and functionalities of a subset of Systemd, focusing on the interfaces required by CARBON-AWARE SYSTEMD TIMERS. The core Systemd manager operates most Systemd components, in particular Systemd timers and the DBus. Systemd timers are a Systemd-internal representation of timed event. The DBus facilitates communication between Systemd components and services.

2.1 Carbon-Awareness Terminology

To establish the goal of CARBON-AWARE SYSTEMD TIMERS requires a definition of carbon awareness. I follow the terms and definitions used by the Green Software Foundation for carbon awareness. First, carbon awareness is an attribute of software, in this case Systemd timers, to respond to the carbon intensity of used resources. Here, carbon measures the carbon-dioxide (CO_2) equivalent of emitted greenhouse gasses, not just carbon emissions. Emissions are then divided into two categories, embodied carbon and operational carbon. Embodied carbon represents the emissions caused during creation and disposal of a piece of hardware. CARBON-AWARE SYSTEMD TIMERS achieve carbon awareness by temporal shifting of workloads. For most resources this means, no impact on embodied carbon is possible. For example the embodied carbon of a CPU executing a workload, does not change depending on when the workload is executed. Instead, I focus on operational carbon, the emissions caused in production of the electricity used to run a workload. To obtain the operational carbon (O) of a workload, its electricity consumption (E) is multiplied with an estimate of regional carbon-intensity (I) of grid electricity, as seen in Equation (2.1).

$$O_{workload} = E_{workload} \cdot I \tag{2.1}$$

[3]Software Carbon Intensity (SCI) Specification [Gre24].

2.2 Systemd

Systemd[4] is a widespread system and service management tool for Linux. In this section I present the general structure of Systemd and in particular the Systemd core manager, timers and DBus.

Figure 2.1 illustrates the structure of Systemd's service management. Systemd operates on a collection of services. A service can be any program, e.g., a web server, backup tool or software updater. Listing 2.1 shows an example definition of a Systemd service. Services may be started at boot time, manually, via external tools or by means of a Systemd timer. Systemd timers and services are, along with various other types, defined by INI files and collectively called units. Systemd timers can specify when a service should be started in terms of either wall clock time, as shown in Listing 2.2, or a duration since a previous event, as seen in Listing 2.3. Each active Systemd timer is represented internal to the manager by an event timer, which I present in detail in Section 2.2.2. Communication between Systemd services, external tools, and core components can be facilitated via DBus. For example `systemctl` is mostly a wrapper around the DBus interface of the manager and `systemd-resolved` can resolve DNS queries submitted via DBus. There may be several DBus instances within one system. The instance required by CARBON-AWARE SYSTEMD TIMERS is the system bus, where the managers DBus interface is available. The manager receives all messages on the system bus and forwards them to their destination. Event timers and DBus messages among others constitute the events processed by the managers main function, the event loop.

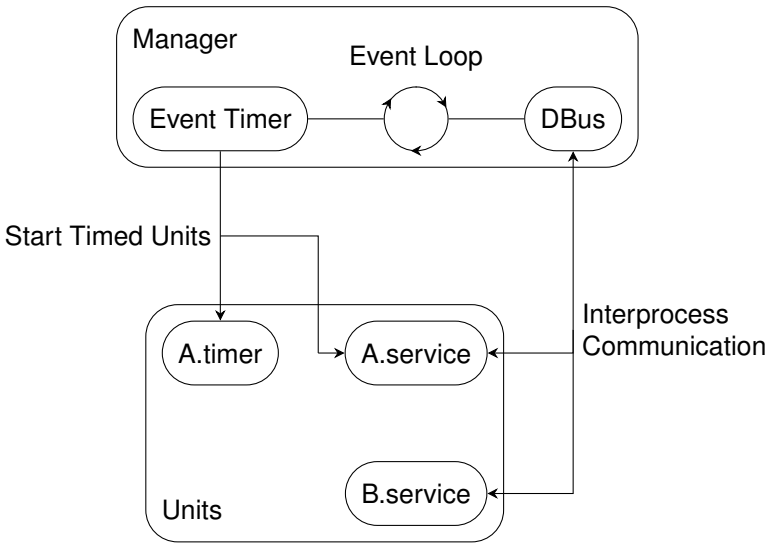

Figure 2.1 – General structure of Systemd, showing the interaction of the Systemd manager with Systemd timers, as well as communication between services via DBus.

```
1 [Service]
2 ExecStart=/usr/sbin/nginx
```

Listing 2.1 – Definition of a Systemd service starting the web server nginx.

[4]Systemd: System and Service Manager [sys24].

```
[Timer]
OnCalendar=*-*-* *:00:00
AccuracySec=1min
Unit=A.service
```

Listing 2.2 – Definition of a Systemd timer triggering A.service every full hour.

```
[Timer]
OnBootSec=0
OnUnitActiveSec=1h
Unit=B.service
```

Listing 2.3 – Definition of a Systemd timer triggering B.service at one-hour intervals from system boot.

2.2.1 Core Manager

The Systemd core manager is the central core of Systemd service management. It processes all internal events, in particular event timers and DBus messages on the system bus. Listing 2.4 shows a simplified version of the manager's event loop, using the epoll interface of the underlying operating system to wait for expiry of any internal event. Expired events are then taken from a pending queue and their respective callbacks are executed. In case of an event timer representing a Systemd timer, the callback starts the service associated with the Systemd timer, possibly along with any dependencies. In case of a DBus message, the callback forwards the message to the receiving bus participant.

```
int manager_loop(Manager *m) {
  while (m->objective == MANAGER_OK) {
    epoll_wait(m->event);
    e = prioq_peek(m->event->pending);
    e->callback();
  }
}
```

Listing 2.4 – Simplified implementation of the manager event loop. For each incoming event a callback is executed, until the objective signals the manager to shut down.

2.2.2 Time-Based Events (Timers)

I distinguish two kinds of timers. First a Systemd timer is an INI file specifying when a service should be executed. Second an event timer is the internal data structure representing the time of the next elapse of an active Systemd timer or any Systemd internal timed event. The manager (Section 2.2.1) multiplexes all of its event timers onto the timers made available by the underlying operating system.

The internal representation of an event timer (Listing 2.5) consists of a callback function and two microsecond values. The `callback` function typically starts an associated Systemd service. The first microsecond value, `next`, is the timestamp, at which this timer should expire at the earliest. The second value specifies an `accuracy`. The timer should expire at the latest `accuracy` microseconds after the *earliest* timestamp. Notably an event timer makes no guarantees on when any actions

associated with it conclude. In case of a Systemd timer, its associated service will not start before the `next` timestamp, but may start even after the `accuracy` interval. The event timer makes no consideration of how long an associated Systemd service might run. The accuracy value of an event timer can be specified by a Systemd timer via the AccuracySec setting as seen in Listing 2.2. The purpose of the accuracy value is to bundle multiple event timers that would expire in short succession. Instead, Systemd expires event timers within each other's accuracy interval at the same time, in order to reduce the number of interrupts.

```
struct {
  sd_event_time_handler_t callback;
  usec_t next, accuracy;
} time;
```

Listing 2.5 – Excerpt of the data structure used to represent event timers in Systemd.

2.2.3 Inter-Process Communication (DBus)

DBus is an Inter-Process Communication (IPC) Bus. The system bus is an instance of DBus operated by the manager. The system bus is used for communication with Systemd services, as well as accessing specific Systemd-internal data and methods.

Listing 2.6 shows the definition of an example DBus method. `ListUnitsByNames` exposes internal information of units from the Systemd core to participants of the system bus. The method receives unit names as an array of strings and returns an array of structs containing various information about matching units. `method_list_units_by_names` is the internal function implementing this method. The last parameter to `SD_BUS_METHOD_WITH_ARGS` is a bitmap specifying additional properties of this method. Most DBus methods exposed by the manager only use the `SD_BUS_VTABLE_UNPRIVILEGED` flag to disable basic access control and instead defer that to e.g., polkit [5].

```
SD_BUS_METHOD_WITH_ARGS("ListUnitsByNames",
  SD_BUS_ARGS("as", names),
  SD_BUS_RESULT("a(ssssssouso)", units),
  method_list_units_by_names,
  SD_BUS_VTABLE_UNPRIVILEGED)
```

Listing 2.6 – Example DBus method definition, receiving a string list of names and returning various information on matching units for each name.

```
sd_bus_call_method(system_bus,
  "org.freedesktop.systemd1",          // destination
  "/org/freedesktop/systemd1",         // path
  "org.freedesktop.systemd1.Manager", // interface
  "ListUnitsByNames",                  // method name
  &error,
  &reply,
  "as",                                // method parameter types
  names);                              // method parameters
```

Listing 2.7 – Example DBus method call, retrieving information on the units listed in the names variable.

[5] polkit is a toolkit for defining and handling authorizations: `https://github.com/polkit-org/polkit`

Listing 2.7 shows an example call to this method. The call requires a connection to the `system_bus` and the following parameters to `bus_call_method`:

- `destination` is a unique identifier of the method provider on this bus
- `path` would allow the destination to export methods in different contexts. Most Systemd services do not make use of this DBus feature and instead always use a path that differs from the destination string only in the separator character.
- `interface` is the name of a collection of methods, which allows different bus participants to provide the same methods, while the manager enforces compatibility of these method signatures.
- `Name` of the method
- Preallocated storage for an `error` and a `reply`
- The parameters to the method and their types

The call will block until a reply or an error is received. Methods can also be called asynchronously, in which case a reply can be received via callback.

3 DESIGN

Current implementations of carbon-aware workload management focus on datacenters. CARBON-AWARE SYSTEMD TIMERS bring carbon awareness to single-machine scenarios, i.e., consumer machines and private or corporate non-datacenter servers. These single-machine scenarios have several relevant implications.

- I can not move workloads to different machines, which denies the benefits of power modeling and geographic shifting. Only temporal shifting remains as an option in single-machine scenarios.
- Workloads in these scenarios typically do not come with strict deadlines and resource requirements.
- Any ahead-of-time scheduling may be disrupted by interactive or manually triggered workloads.

In Section 3.1 I identify services suitable for temporal shifting [Wie+21] from the Debian package repositories. With these packages in mind I expand Systemd functionality to apply delays based on external carbon intensity prediction and resource usage tracking. In Section 3.2 I present a modular approach to add the required functionality to Systemd. The requirements for the scheduling algorithm for Systemd timers differ significantly from those applicable in a datacenter environment. I examine these differences in Section 3.3. Concluding the design I present an overview of its benefits in Section 3.4.

3.1 Use-Case Analysis

As a base use case for carbon awareness in Systemd timers I examine packages containing Systemd timers in the main section of the official Debian 12 repositories [6]. Table A.1 shows all 84 packages containing Systemd timer unit files. For each package I sort the task associated with its Systemd timers into the following categories:

- Backup: Create or manage backup copies of data.
- Data Checks: Perform some sort of maintenance or verification on local data, e.g., file system maintenance or cache cleanup.
- Notifications: Notify a user of changes or required actions.
- Service Restart: Terminate and restart another service.

[6] `https://packages.debian.org/bookworm/allpackages`

- Statistics: Collect data changing over time.
- Task Scheduling: Perform scheduling of tasks outside Systemd's timer scheduling.
- Updates: Adapt local data from a remote source or vice versa.

I roughly estimate whether the execution duration of the tasks associated with each timer can be expected to be above or below several minutes. Actual execution duration however, may vary a lot for all tasks depending on individual configuration choices. Lastly I state for each package at what frequency their timer elapse and whether the timers operate on wall clock time or monotonic system uptime.

Not all categories of tasks are equally suited for carbon-aware delays. Notification, service restart, statistic, and task scheduling tasks are typically short-running tasks, therefore consuming little electricity and offering little potential to reduce carbon emissions. Tasks operating at a frequency close to or higher than the granularity of the carbon-prediction data can not be meaningfully delayed based on that carbon prediction, as the prediction remains constant within the task's possible start interval. Timers operating on monotonic time are less suited for carbon awareness, since carbon intensity varies based on wall time, not individual system uptime. However, note that for most of these timers monotonic versions could be substituted with wall clock versions without significant impact on their task's functionality.

When excluding timers with too high frequency or incompatible clock types, 49 packages remain. When further excluding tasks with an execution duration likely below several minutes, the list of Debian packages boils down to Table 3.1. The remaining 10 tasks all perform file-system or database maintenance, software updates or backups. All of these can be delayed within their frequency interval, without significant impacts on their functionality.

3.2 System and Service Management (Systemd)

In order to delay Systemd timers in a carbon-aware manner, I define DBus interfaces to modify running timers, retrieve carbon intensity predictions, and collect execution durations of relevant services. In Sections 3.2.1 to 3.2.3, I outline three Systemd services connecting these interfaces. The RESOURCE TRACKER stores resource usage information of other services. The CARBON PREDICTOR service retrieves carbon intensity predictions from internal or external sources. The CARBON MANAGER modifies Systemd timers based on data from the previous two services.

Package	Task	Frequency
apt	updates	Day
borgmatic	backup	Day
btrbk	backup	Day
btrfsmaintenance	data checks	Month
e2fsprogs	data checks	Week
ocsinventory-agent	data checks+updates	Day
podman	updates	Day
postgresql-common	backup+data checks	Day-Week
util-linux	data checks	Week
zfsutils-linux	data checks	Week-Month

Table 3.1 – Debian packages from Table A.1 reduced to packages well suited for carbon aware Systemd timers.

3.2.1 Resource Tracker

Systemd already maintains the execution duration and CPU consumption of each service's most recent execution. In order to estimate resource usage of future executions the RESOURCE TRACKER stores and averages data of multiple previous executions. Carbon-aware scheduling of a service may alter resource contention between the delayed service and other service now running concurrently. To account for changing resource contention conditions, the Systemd manager continuously updates the data collected on carbon-aware-delayed services. The RESOURCE TRACKER implements a DBus interface, allowing the Systemd manager to add duration and CPU consumption of a services execution and allowing the CARBON MANAGER to retrieve averages of a services past resource-usage data.

3.2.2 Carbon Predictor

For carbon intensity prediction I rely on existing data or external services. The CARBON PREDICTOR service may retrieve data from online services such as Electricity Maps or operate on static data of hourly, daily or more coarse fluctuations in past carbon intensity values. It offers a DBus interface to retrieve predictions in a specified target interval in the future. These predictions are unified to a fixed granularity independent of the data source to simplify implementation of modules interacting with this interface.

3.2.3 Carbon Manager

The CARBON MANAGER combines data from the RESOURCE TRACKER and the CARBON PREDICTOR service to modify running Systemd timers and optimize their carbon emissions. Systemd timers are operated by the Systemd manager, which I expand by a DBus interface allow the CARBON MANAGER to change the remaining time of a specified timer.

3.3 Carbon-Aware Scheduling

There are already multiple approaches to the optimization problem of when and where to execute tasks across datacenters for minimal carbon emissions as described by Yang et al. [YHF22] and Radovanović et al. [Rad+22]. However, workloads in a datacenter setting are typically associated with SLAs specifying deadlines and minimally required resources, while the use cases listed in Section 3.1 require neither significant minimal resources nor fixed deadlines. This allows CARBON-AWARE SYSTEMD TIMERS to deploy a simple scheduling algorithm, without violation resource or deadline constraints, while still achieving the carbon emission reductions shown in Chapter 5. To illustrate the benefits of workload flexibility I examine possible scheduling decisions for two example tasks with and without flexible resource requirements. Figure 3.1 shows an example timeline of electricity grid carbon intensity varying between 20 and 110 $\frac{\mathrm{gCO_2}}{\mathrm{kWh}}$ across the course of 6 hours. Figure 3.2 shows two tasks A and B each consuming 75% of this host's computational capacity at 75 W for 2 hours. If both tasks are guaranteed access to 75% CPU time for their full execution duration, the shown scheduling is optimal for total operational carbon emissions at 2.25 $\mathrm{gCO_2}$ for task A, 9 $\mathrm{gCO_2}$ for task B and a total of 11.25 $\mathrm{gCO_2}$ as shown in Equations (3.1) to (3.3). Figure 3.3 shows an optimal scheduling, if the tasks have no requirement for minimal guaranteed CPU access. The tasks contest for CPU time resulting in a longer execution time of 3 h for both, but in return both tasks fit into the low carbon intensity interval between $1-4$ h. As a result the carbon emissions

are reduced to 2.25 gCO_2 for both tasks at a total of 4.5 gCO_2 as seen in Equations (3.4) and (3.5). In this example the flexible requirements of the tasks allowed a 60% reduction in carbon emissions.

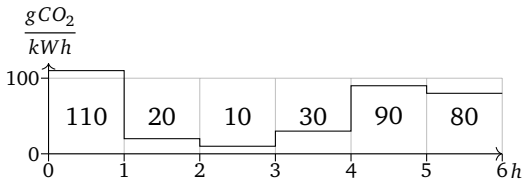

Figure 3.1 – Example carbon intensity timeline for 6 hours, ranging from 10 to 110 gCO2/kWh.

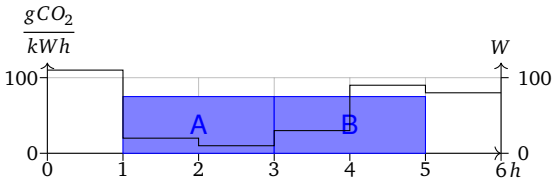

Figure 3.2 – Optimal scheduling for two services (*A* and *B*), if each have a fix requirement of 75% CPU capacity for 2 hours.

$$75\,\mathrm{W} \cdot \left(20\,\frac{\mathrm{gCO_2}}{\mathrm{kWh}} \cdot 1\,\mathrm{h} + 10\,\frac{\mathrm{gCO_2}}{\mathrm{kWh}} \cdot 1\,\mathrm{h}\right) = 2.25\,\mathrm{gCO_2} \tag{3.1}$$

$$75\,\mathrm{W} \cdot \left(30\,\frac{\mathrm{gCO_2}}{\mathrm{kWh}} \cdot 1\,\mathrm{h} + 90\,\frac{\mathrm{gCO_2}}{\mathrm{kWh}} \cdot 1\,\mathrm{h}\right) = 9\,\mathrm{gCO_2} \tag{3.2}$$

$$2.25\,\mathrm{gCO_2} + 9\,\mathrm{gCO_2} = 11.25\,\mathrm{gCO_2} \tag{3.3}$$

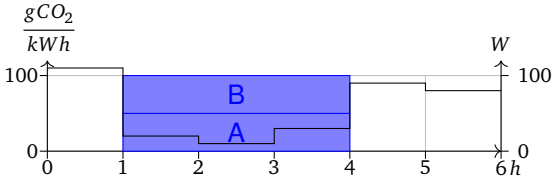

Figure 3.3 – Optimal scheduling for two services (*A* and *B*), if each have a flexible requirement of the equivalent of 75% CPU capacity across 2 hours, but allow stretching the execution duration for reduced CPU usage.

$$50\,\mathrm{W} \cdot \left(20\,\frac{\mathrm{gCO_2}}{\mathrm{kWh}} \cdot 1\,\mathrm{h} + 10\,\frac{\mathrm{gCO_2}}{\mathrm{kWh}} \cdot 1\,\mathrm{h} + 30\,\frac{\mathrm{gCO_2}}{\mathrm{kWh}} \cdot 1\,\mathrm{h}\right) = 2.25\,\mathrm{gCO_2} \tag{3.4}$$

$$2.25\,\mathrm{gCO_2} + 2.25\,\mathrm{gCO_2} = 4.5\,\mathrm{gCO_2} \tag{3.5}$$

Furthermore, CARBON-AWARE SYSTEMD TIMERS, especially in the setting of consumer machines, must be able to function in the presence of user controlled interactive tasks. This prevents reliable prediction of resource availability. Given the flexibility of tasks in my use case and the uncertainty of resource availability, I chose a simplistic scheduling algorithm, which greedily places each task in an interval of minimal carbon intensity. Resource contention is only taken into account through feedback of the RESOURCE TRACKER observing longer execution durations in case of repeated contention.

3.4 Design Summary

The modular design of CARBON-AWARE SYSTEMD TIMERS eases adaptation to different use cases. Implementation of each module can be replaced at runtime, without disrupting operation of the underlying system. Additionally, adapting or creating new implementations does not in itself require changes to other modules or the system manager.

The RESOURCE TRACKER collects resource usage statistics for carbon-aware timers. It can be disabled for data-protection concerns or extended to cover more resources such as network or disc usage in order to accurately predict resource contention.

The CARBON PREDICTOR provides carbon-intensity forecasts through a unified interface. It can be run in an offline variant, to avoid dependency on online services. Alternatively, it can be adapted to any available source of carbon-intensity data, including statistics for locally generated electricity, such as an on-site photovoltaic farm. The abstraction from regional carbon-intensity data through the CARBON PREDICTOR allows carbon awareness for suitable services, without the need for region-specific configuration of individual services.

The CARBON MANAGER combines the data from the other two modules into a simple scheduling algorithm to reduce carbon emissions. Should a use case require more intricate scheduling decision, the CARBON MANAGER module can be easily replaced by a more complex implementation.

4 IMPLEMENTATION

CARBON-AWARE SYSTEMD TIMERS apply delays to workloads based on grid carbon intensity and characteristics of each workload. To achieve this, the implementation is structured into 3 separate interacting modules. The RESOURCE TRACKER collects resource usage information from past service executions. The CARBON PREDICTOR retrieves online carbon intensity information for the regional electricity grid and provides the intensity data through a unified interface. The CARBON MANAGER delays Systemd timers according to the combined data of the previous two modules. This chapter focuses on the interfaces and interactions between these modules and the Systemd core. Finally, Section 4.4 illustrates the ease of deploying CARBON-AWARE SYSTEMD TIMERS to a Debian system.

4.1 Resource-Usage Tracking

In order to avoid the execution of a service associated with a carbon-aware timer extending into time of high carbon intensity, it is necessary to estimate the execution duration of the workload. The RESOURCE TRACKER solves this by aggregating mean execution durations from past executions. Systemd already collects the duration of the most recent execution, which can be viewed for a currently running service with the `systemctl status <service>` command, see Listing 4.1.

```
1 > systemctl status ssh.service
2      Active: active (running) since Tue 2024-09-03; 1 month 20 days ago
3         CPU: 78ms
```

Listing 4.1 – Excerpt of Systemd status information on a currently running service, showing running duration and CPU usage.

To allow the RESOURCE TRACKER to collect this information from the Systemd core and serve aggregated means to the CARBON MANAGER, I define the `org.freedesktop.resourcetracker` DBus interface, shown in Listing 4.2. The interface defines two methods. `AddResourceUsage` stores a new data set associated with a Systemd service by its `service_name`. `GetAverageResourceUsage` retrieves the means of previously stored data for the named service. For details on the remaining arguments of `SD_BUS_METHOD_WITH_ARGS`, see Section 2.2.3.

Finally, to send data sets for past service executions to the RESOURCE TRACKER, I modify the Systemd core to call `AddResourceUsage` upon completion of a service. Note that, the CPU usage collected by Systemd stems from the Linux kernels Control Group[7] interface and does not reflect changes in CPU frequency. The electricity consumption of CPUs varies significantly with CPU frequency [Ibr+16] and as such, would be of interest for carbon-aware service management.

[7] Control Group v2 [Lin15].

In conclusion, the RESOURCE TRACKER expands Systemd's functionality, by storing resource-usage data across several past service executions and offering a DBUS interface to retrieve means of this data.

```
SD_BUS_METHOD_WITH_ARGS("AddResourceUsage",
                        SD_BUS_ARGS("s", service_name, "t", duration, "t", ↘
                            cpu_usage),
                        SD_BUS_NO_RESULT,
                        add_resource_usage,
                        SD_BUS_VTABLE_UNPRIVILEGED),
SD_BUS_METHOD_WITH_ARGS("GetAverageResourceUsage",
                        SD_BUS_ARGS("s", service_name),
                        SD_BUS_RESULT("t", avg_duration, "t", avg_cpu_usage),
                        get_average_resource_usage,
                        SD_BUS_VTABLE_UNPRIVILEGED),
```

Listing 4.2 – DBus interface implemented by the RESOURCE TRACKER, to collect and provide resource usage information on Systemd services.

4.2 Carbon-Intensity Forecast Provision

Carbon awareness requires knowledge about the carbon intensity of consumed electricity. The CARBON PREDICTOR supplies this information through `org.freedesktop.carbonpredictor`, a newly defined DBUS interface, and provides abstraction from the specific source of carbon-intensity data. Listing 4.3 shows the interface providing the data via the `GetCarbonIntensity` method. Given a desired time interval by `from` and `to` timestamps, the method returns a half-hour aligned array of forecast carbon-intensity values. The prediction may be shorter than the requested interval, if the data source can not supply sufficient data.

```
SD_BUS_METHOD_WITH_ARGS("GetCarbonIntensity",
                        SD_BUS_ARGS("t", from, "t", to),
                        SD_BUS_RESULT("t", prediction_from, "ad", prediction),
                        get_carbon_intensity,
                        SD_BUS_VTABLE_UNPRIVILEGED),
```

Listing 4.3 – DBus interface implemented by the CARBON PREDICTOR, to provide carbon intensity information through a unified abstraction from the specific source.

A suitable data source would be the carbon-intensity forecast by Electricity Maps, however their data is not freely available[8]. As a substitute, this implementation provides two alternative data sources. The STATIC CARBON PREDICTOR operates on static table of carbon intensity values and the ENTSO-E PRICE PREDICTOR operates on day-ahead electricity prices retrieved from ENTSO-E's RESTful API[9].

[8] They did not respond to my request for research access to their carbon-intensity forecasts.
[9] ENTSO-E Transparency Platform RESTful API [ENT24b].

The carbon-intensity data used by the STATIC CARBON PREDICTOR is a table of means for every half hour of a week. The means are taken across the year 2023 from the historic data available from Electricity Maps. This table accounts for some daily and weekly fluctuations, investigated by [Wie+21], such as:

- Availability of solar power during midday.
- Reduced electricity demand at night.
- Reduced electricity demand during weekends

While this method is significantly less accurate than actual carbon forecasts, it does provide the basic patterns of regional carbon intensity. In summary, the STATIC CARBON PREDICTOR allows CARBON-AWARE SYSTEMD TIMERS to exploit regional patterns in carbon intensity, without requiring access to online services.

The ENTSO-E PRICE PREDICTOR implements the carbon predictor interface and provides live online prediction values. However, its data source are not carbon-intensity values, but instead day-ahead electricity prices. When using this predictor, CARBON-AWARE SYSTEMD TIMERS will optimize monetary cost instead of carbon emissions. This demonstrates the flexibility of CARBON-AWARE SYSTEMD TIMERS in allowing the use of arbitrary input metrics. This predictor further serves as a sample implementation for use of an online data source. It can easily be adapted to a carbon-intensity forecast API, if access to such an API can be acquired. In conclusion, the ENTSO-E PRICE PREDICTOR serves to demonstrate operation on an online data source.

In summary, the CARBON PREDICTOR provides carbon-intensity forecasts, while abstracting from the source of data. This abstraction has two noteworthy benefits. The source of data may be replaced by another without adjusting other parts of Systemd, CARBON-AWARE SYSTEMD TIMERS, or individual workload services. Additionally, the abstraction allows exploitation of regional carbon-intensity patterns, without requiring individual services to have region-specific configuration.

4.3 Carbon-Aware Timer Management

With the data from the RESOURCE TRACKER and CARBON PREDICTOR in place, the last step is to combine this data into scheduling decisions. The CARBON MANAGER applies delays to Systemd timers in order to reduce operational carbon of their associated Systemd services. To allow modification of Systemd timers by a non-core service, such as the CARBON MANAGER, I expand the DBus interface of the Systemd manager by the methods listed in Listing 4.4. Note, `SD_BUS_PROPERTY` is just a convenience alternative to a method for retrieval of a value or set of values. `CarbonTimers` provides the CARBON MANAGER with the list of Systemd timer, marked as suitable for carbon-aware delays. `SetCarbonTimer` modifies the next expiration time of the given Systemd timer. Finally, `SetNextCarbonTimerSchedule` informs the Systemd manager when the CARBON MANAGER will run next.

Given the large intervals at which carbon-aware delays need to operate, it is sufficient for the CARBON MANAGER to run at the granularity of the carbon-intensity forecast, i.e., every half hour. If the Systemd manager encounters any newly established timers, which expire before the next run of the CARBON MANAGER, timer scheduling can fall back to non-carbon-aware behavior for this individual timer expiration.

```
SD_BUS_PROPERTY("CarbonTimers", "a(stt)", property_get_carbon_timers, 0, 0),
SD_BUS_METHOD_WITH_ARGS("SetCarbonTimer",
                        SD_BUS_ARGS("s", name, "t", next_elapse_realtime),
                        SD_BUS_NO_RESULT,
                        method_set_carbon_timer,
                        SD_BUS_VTABLE_UNPRIVILEGED),
SD_BUS_METHOD_WITH_ARGS("SetNextCarbonTimerSchedule",
                        SD_BUS_ARGS("t", next_carbon_timer_schedule),
                        SD_BUS_NO_RESULT,
                        method_set_next_carbon_timer_schedule,
                        SD_BUS_VTABLE_UNPRIVILEGED),
```

Listing 4.4 – DBus interface extension to the Systemd manager, to support timer modification.

The interval in which the CARBON MANAGER may delay a timer, must be configured for each suitable timer. A maximum interval can be assumed to be the frequency of the timer. However, it depends on the individual service, whether delays of this maximum length can be applied without interference with the service's functionality. A further complication stems from the behavior of Systemd timer. If a Systemd timer expires, while its associated service is running, no action is taken. If the CARBON MANAGER delays a service just short of the maximum interval, the following expiration might occur, while the service is still running. This would result in the service being executed fewer times than expected. If this is an issue, the delay interval must be configured short enough to avoid service execution overlapping with a following interval.

In summary, the CARBON MANAGER combines data from the RESOURCE TRACKER and CARBON PREDICTOR to delay workloads into times of low carbon-intensity. Systemd timers must be marked as suitable for carbon awareness, by configuring an interval in which the timer may be delayed.

```
[Timer]
OnCalendar=daily
CarbonSec=22h
```

Listing 4.5 – Definition of a Systemd timer made carbon-aware by CARBON-AWARE SYSTEMD TIMERS.

4.4 Deployment and Configuration Requirements

CARBON-AWARE SYSTEMD TIMERS are deployable as a series of Debian packages, built on Systemd version 252. The changes to the Systemd core are available in the packages `libsystemd-shared`, `libsystemd0`, and `systemd`. Each module is an individual package. To deploy CARBON-AWARE SYSTEMD TIMERS, install the 3 core packages, `systemd-resource-trackerd`, `systemd-carbon-manager`, and either `systemd-carbon-prediction-staticd` or `systemd-carbon-prediction-entsoed`. The ENTSO-E predictor requires an API Access Token stored in the file `/etc/carbon-prediction-entsoe/secret`. Any suitable timer can now be made carbon-aware by adding a delay interval to its unit file via the `CarbonSec` option, as seen in Listing 4.5. For typical use cases, such as those listed in Section 3.1, the delay interval configuration can be done by package maintainers, requiring no further changes by machine administrators.

5 EVALUATION

To show the effectiveness of CARBON-AWARE SYSTEMD TIMERS, I measure the savings of subjecting a daily task to carbon-aware delays for one week. Additionally, I simulate one of the use cases listed in Section 3.1 across the full year 2023 to illustrate the limits and possibilities of single-machine workload time shifting in the current German grid electricity mix.

5.1 Setup

My testing setup consists of a clean Debian Bookworm installation in a QEMU[10] virtual machine on an Intel® Core™ i5-8400 host with KVM[11] enabled. I deploy CARBON-AWARE SYSTEMD TIMERS to the virtual machine and measure energy consumption using linux-perf[12] to retrieve the Intel® Running Average Power Limit (RAPL)[13] energy counter for the package domain at 5-minute intervals. The package domain measures energy consumption of the entire CPU socket. Consumption by other hardware components, such as hard discs, network cards or the motherboard is not included in these measurements.

Electricity Maps carbon-intensity forecast was not available to me as input to the real-time evaluation, because Electricity Maps did not respond to my inquiries for research access to their forecast data. As a substitute I schedule workloads according to day-ahead electricity prices freely available from ENTSO-E[14]. Note that day-ahead electricity prices are not a forecast, but instead businesses may commit to purchasing electricity one day ahead at these prices. Day-ahead prices are no sufficient substitute, when minimizing carbon emissions. However, they serve as an example to demonstrate CARBON-AWARE SYSTEMD TIMERS' ability to minimize workload cost according to an input metric. I choose to use price and carbon-intensity data for Germany in this evaluation, as Wiesner et al. show Germany to have the largest daily variations in carbon intensity [Wie+21].

I evaluate three benchmarks and compare the cost of a test workload scheduled in each benchmark with and without carbon awareness. First, the ONLINE TIMER SCHEDULING benchmark observes a live run of CARBON-AWARE SYSTEMD TIMERS for 7 days in real time, using freely-available day-ahead prices as input metric. Second, the EMISSION REDUCTIONS FOR REAL-WORLD BACKUP TASK benchmark simulates a daily workload across a full year, using historic carbon-intensity data, as well as day-ahead prices, of the year 2023. Finally, the OFFLINE TIMER SCHEDULING benchmark shares

[10] QEMU, a fast and portable dynamic translator [Bel05].
[11] Kernel-Based Virtual Machine [Kiv+07].
[12] perf: Linux profiling with performance counters [Lin24].
[13] Running Average Power Limit [Kha+18].
[14] ENTSO-E Transparency Platform [ENT24a].

the scenario of the EMISSION REDUCTIONS FOR REAL-WORLD BACKUP TASK, but evaluates the STATIC CARBON PREDICTOR, compared to online alternatives.

The ONLINE TIMER SCHEDULING benchmark evaluates the effectivity of CARBON-AWARE SYSTEMD TIMERS in real time with day-ahead prices retrieved live from the ENTSO-E API for 7 days from the 8th through the 14th September 2024. Figure 5.1 shows the day-ahead prices for Germany within this time period. The workload for this benchmark consists of 1 CPU-hour executed daily, between 2:00 and 24:00. The 2 hour gap in this start interval is the result of a limitation of Systemd timers, see Section 4.3. The interval must ensure that each execution does not overlap with the following interval. I compare this benchmark with a simulation of a non-carbon-aware scheduling. The simulation combines samples of electricity consumption while idling and during workload execution. The idle sample is a measurement of the electricity consumption of the evaluation system with no test workloads scheduled seen in Figure 5.2. The workload sample shown in Figure 5.3 is a measurement of one execution of the 1-CPU-hour workload. The non-carbon-aware simulation schedules the workload at the same time of day each day. The time of day is chosen to yield the lowest carbon intensity across the observed week.

The EMISSION REDUCTIONS FOR REAL-WORLD BACKUP TASK benchmark is a simulation, operating on historic carbon intensity data, obtained from Electricity Maps, for Germany across the full year 2023. The workload for this benchmark is a run of the Borg[15] backup tool invoked by the Borgmatic package from Table 3.1 on a directory containing one 50 GB file of random data and a checkout of the Linux kernel repository. The electricity consumption of this workload used in the simulation is shown in Figure 5.4. The carbon-intensity data used as input metric for the simulated scheduling is shown in Figure 5.5. I compare this carbon-aware simulation with 4 further simulations. The first other simulation operates on day-ahead electricity prices in order to evaluate the suitability of day-ahead prices for optimization of carbon emissions. The day-ahead prices for the year 2023 are shown in Figure 5.6. Following are three simulations, scheduling the workload at fixed times of day. First, the workload is scheduled at 00:00 every day, which is the current behavior of the Borgmatic package. Second, the daily start time is 13:00, yielding the lowest emissions of any fixed starting time for the year 2023. Third, starting the workload at 20:00 each day results in the highest carbon emissions for any unaware timer.

Finally, the OFFLINE TIMER SCHEDULING benchmark sets the workload timer according to the STATIC CARBON PREDICTOR from Section 4.2. The benchmark shares the same environment, as the EMISSION REDUCTIONS FOR REAL-WORLD BACKUP TASK benchmark, but operates without depending on any online services.

[15] Deduplicating archiver with compression and encryption [Bor15].

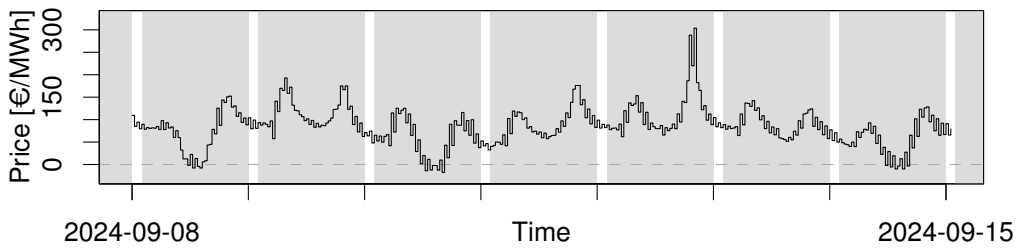

Figure 5.1 – Day-Ahead Electricity Prices for Germany ranging between −17.46 €/MWh and 303.79 €/MWh from September 8th through 15th 2024 at a half-hour resolution

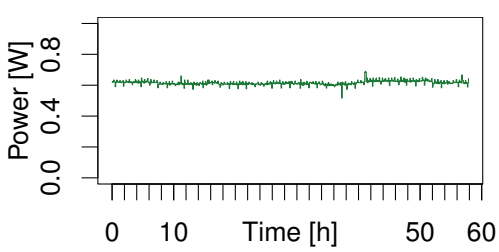

Figure 5.2 – Power consumption of the benchmark host with no workloads, measured for 58 hours, averaging at 0.61 W

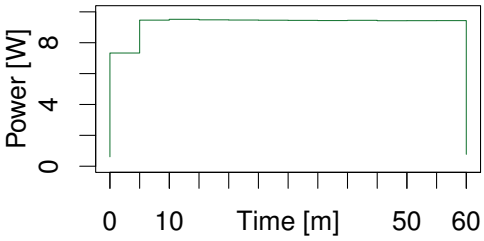

Figure 5.3 – Power consumption of a 1-CPU-hour workload on the benchmark host. Peak electricity consumption is 9.71 W

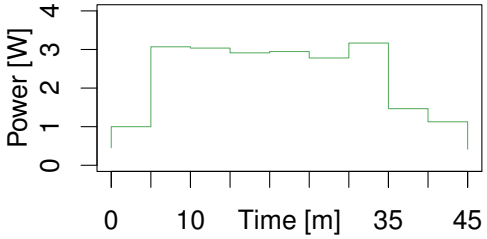

Figure 5.4 – Power consumption of the backup tool Borg executed on a directory with a 50 GB file of random data and a checkout of the Linux kernel repository on the benchmark host. Execution duration is 45 minutes. Peak electricity consumption is 3.37 W

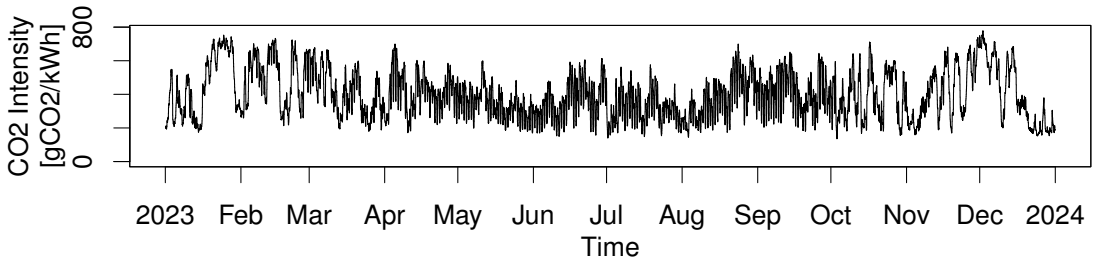

Figure 5.5 – Carbon-Intensity data from Electricity Maps for Germany ranging between 135.48 gCO2/kWh and 778.89 gCO2/kWh for the full year of 2023 at a 1-hour resolution

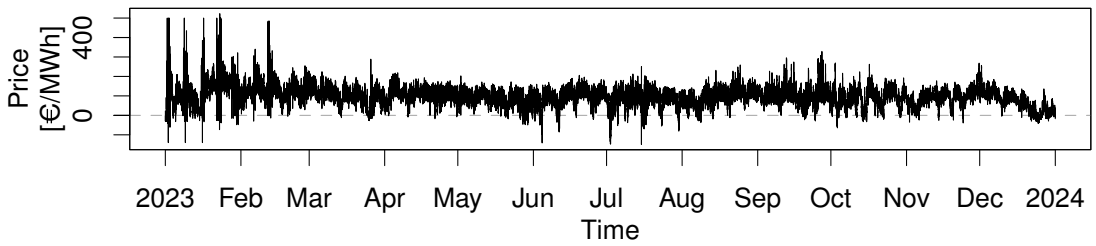

Figure 5.6 – Day-Ahead Electricity Prices for Germany ranging between −149.475 €/MWh and 522.995 €/MWh for the full year of 2023 at a half-hour resolution

5.2 Online Timer Scheduling

The ONLINE TIMER SCHEDULING benchmark operates CARBON-AWARE SYSTEMD TIMERS in a 7-day live online setting. Comparing the mean workload cost of price-aware, as well as non-aware timers, shows the effectivity of CARBON-AWARE SYSTEMD TIMERS in reducing cost according to an input metric, in this case, day-ahead electricity prices.

Scheduling a 1-CPU-hour workload daily in real time for one week, using day-ahead prices as input metric, resulted in the power consumption shown in Figure 5.7. Each peak corresponds to a run of the test workload. All peaks occurred at the lowest price in their respective execution interval. I can further observe, that the idle power consumption P_{idle} drops on the 12th at 14:50 from 0.6 W to 0.4 W. This drop happened because the test host is not a fully isolated environment and is not related to the benchmark.

In order to obtain the cost incurred by the workload according to the input metric, I first extract the electricity consumption $E_{workload}$ of each test workload run, as shown in Equation (5.1). $E_{workload}$ is the product of workload power consumption $P_{workload}$ and workload execution duration $T_{workload}$. $P_{workload}$ is a portion of the total power consumption P_{total}, extracted by removing idle power consumption P_{idle}. I assume P_{idle} to be constant for the duration of each workload execution. Further multiplying $E_{workload}$ by the input metric, Equation (5.2) reveals the mean cost $\overline{Cost}_{workload}$ of each workload run. In the 7 days from the 8th through the 14th September 2024, scheduling a 1-CPU-hour workload by CARBON-AWARE SYSTEMD TIMERS resulted in a mean workload cost of 0.029¢ according to day-ahead electricity prices.

$$E_{workload} = P_{workload} \cdot T_{workload} = (P_{total} - P_{idle}) \cdot T_{workload} \tag{5.1}$$

$$\overline{Cost}_{workload} = mean(E_{workload} \cdot Price) = 0.029¢ \tag{5.2}$$

I compare the cost of this price-aware scheduled workload, with the cost incurred by non-aware scheduling. To choose a fixed daily starting time, I simulate scheduling the 1-CPU-hour with every half hour as starting time. Figure 5.8 show the optimal fixed starting time to be 13:30, when minimizing day-ahead cost within the 7 days from the 8th September 2024. The power consumption for this optimal fixed-time-of-day scheduling is shown in Figure 5.9. Following Equations (5.1) and (5.2) results in a mean workload cost of 0.034¢ for this non-aware scheduling.

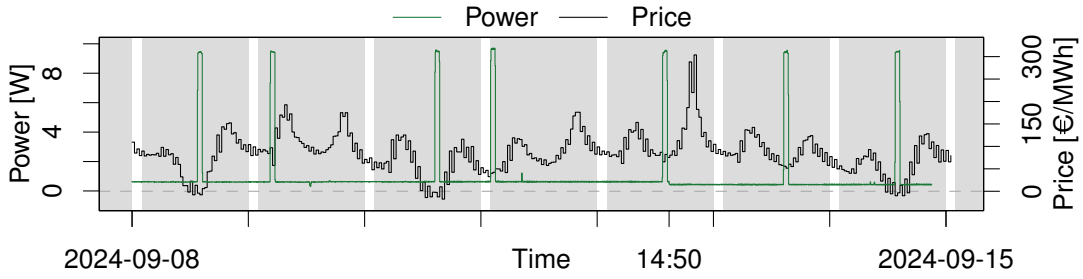

Figure 5.7 – Measured power consumption of the 1-CPU-hour workload. Executed daily from September 8th through 14th. Scheduled by a CARBON-AWARE SYSTEMD TIMER. Overlaid with the day-ahead electricity prices.

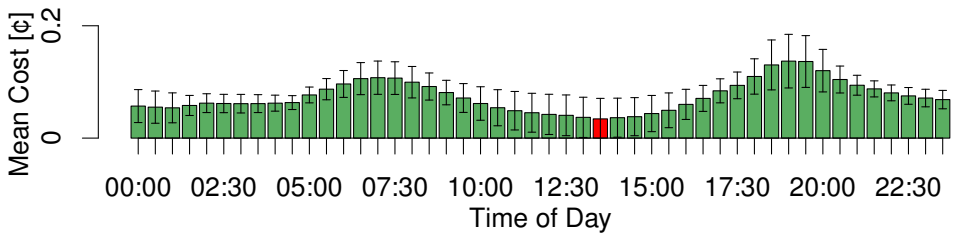

Figure 5.8 – Means of simulated electricity cost following Equations (5.1) and (5.2) for the 1-CPU-hour workload by fixed daily start time from September 8th through 14th. Mean costs range from 0.034 ¢ to 0.137 ¢ with standard deviations between 0.012 ¢ and 0.048 ¢. The lowest cost is reached, when starting the workload at 13:30 each day.

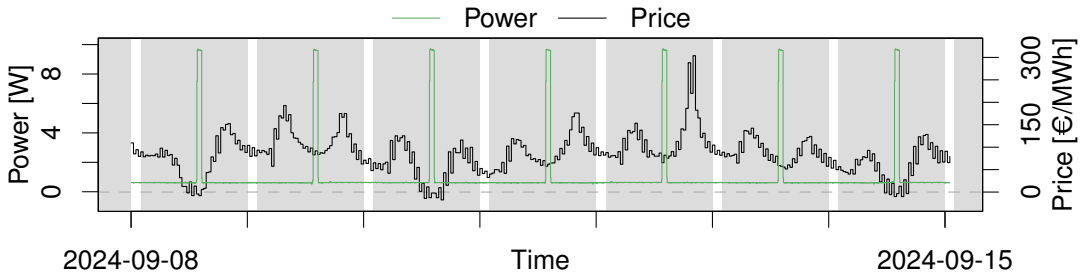

Figure 5.9 – Simulated power consumption of the 1-CPU-hour workload. Executed daily at 13:30 from September 8th through 14th. Overlaid with the day-ahead electricity prices.

The resulting costs for the price-aware scheduling, non-aware scheduling at the optimal and worst time of day, as well as a fixed starting time of 00:00, are listed in Table 5.1. CARBON-AWARE SYSTEMD TIMERS reduce the mean workload cost in this benchmark by at least 17%, from an optimal fixed-time-of-day scheduling, up to 79%, when compared to running the 1-CPU-hour workload at 19:00 each day. Furthermore, Systemd timers, specified as 'daily', are executed at 00:00. Compared to such a daily Systemd timer, CARBON-AWARE SYSTEMD TIMERS achieve a reduction of 50%, according to day-ahead prices in this benchmark.

Scheduling by		Mean Cost [¢]
Optimal Fixed Time	13:30	0.034 ± 0.037
Default Fixed Time	00:00	0.057 ± 0.029
Worst Fixed Time	19:00	0.137 ± 0.048
Price-Aware		0.029 ± 0.033

Table 5.1 – Mean costs per workload execution with 1 standard deviation intervals. Values for fixed daily starting times of the 1-CPU-hour workload from September 8th through 14th compared to scheduling by CARBON-AWARE SYSTEMD TIMERS with day-ahead prices as input metric.

5.3 Emission Reductions for Real-World Backup Task

This benchmark simulates executions of a realistic backup task across the full year 2023, using historic carbon-intensity data, in order to demonstrate the potential for carbon emission reductions offered by CARBON-AWARE SYSTEMD TIMERS. The carbon-intensity data available from Electricity Maps for this benchmark is not a forecast, therefore this benchmark can not account for forecast inaccuracy. Comparing the mean workload costs of carbon-aware, price-aware, and non-aware scheduling reveals two conclusions. First, CARBON-AWARE SYSTEMD TIMERS successfully reduce carbon emissions of suitable workloads in a long term scenario. Second, day-ahead electricity prices are not a suitable replacement for carbon-intensity forecasts, when minimizing carbon emissions.

Mean costs for non-aware scheduling build a baseline, to measure improvements by price- or carbon-aware scheduling. Figure 5.10 shows the mean workload costs for half-hour-aligned unaware scheduling. Similar to the ONLINE TIMER SCHEDULING benchmark, day-ahead prices show local minima in the early morning and early afternoon. The optimal fixed time of day for lowest monetary cost is 13:30. For comparison, the mean workload carbon emissions for unaware scheduling are listed in Figure 5.11. The optimal fixed start time for lowest carbon emissions is 13:00. The nightly local minimum is less pronounced, compared to day-ahead prices. The analysis of German and other grid electricity mixes by Wiesner et al. [Wie+21] shows, the midday minimum on one hand, stems from solar electricity production, which shows in both carbon intensity and day-ahead prices. The early morning minimum on the other hand, is caused by a low in electricity consumption, which shows predominantly in day-ahead prices, while it has less influence on carbon intensity. Both metrics do lead to the same conclusion, that the optimal fixed start time for this test workload is the early afternoon.

In order to choose an optimal interval for carbon-aware scheduling, in which the daily workload is started, I analyze the patterns in carbon intensity of German grid electricity. The day-ahead prices during the ONLINE TIMER SCHEDULING benchmark, follow a fairly regular pattern of local minima in the early morning and early afternoon. In contrast, day-ahead prices, as well as carbon intensity, across the full year 2023 include less regular variations. Figure 5.12 shows how often each time of day has the minimal carbon intensity of the respective day, throughout the year 2023. During most days, the minimal carbon intensity occurs in the early afternoon. However, outliers show minima in carbon intensity at any time of day. To prevent workload execution overlapping with the following start interval, a gap longer than the workload duration is required, i.e. 1 h. Simulating each hour-aligned 23-hour interval, reveals the optimal interval for this benchmark to be between 2:00 and 01:00 the following day.

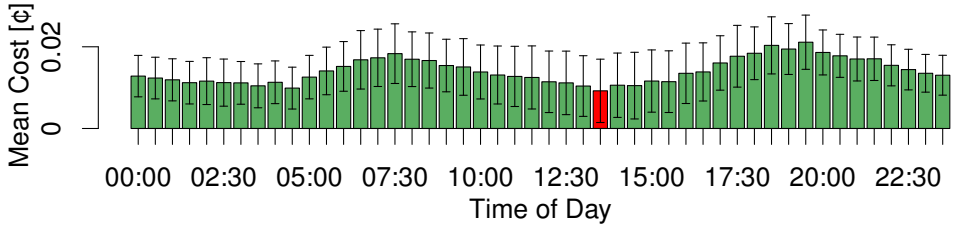

Figure 5.10 – Means of simulated electricity cost per workload execution following Equations (5.1) and (5.2) for the Borg workload by fixed daily start time throughout the year 2023. Mean costs range from 0.009 ¢ to 0.021 ¢ with standard deviations between 0.005 and 0.008. The lowest cost is reached, when starting the workload at 13:30 each day.

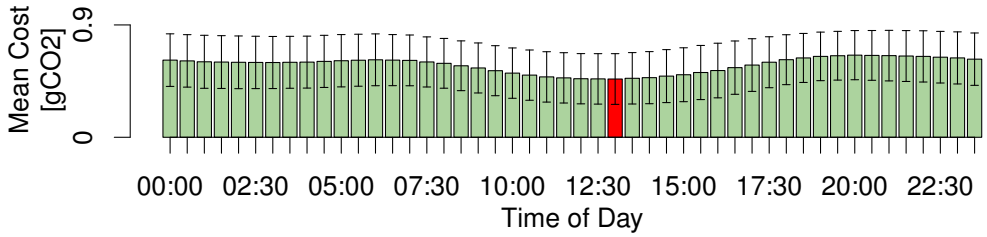

Figure 5.11 – Means of simulated operation carbon per execution of the Borg workload by fixed daily start time throughout the year 2023. Mean emissions range from 0.467 gCO2 to 0.658 gCO2 with standard deviations between 0.194 gCO2 and 0.219 gCO2. The lowest emissions are reached, when starting the workload at 13:00 each day.

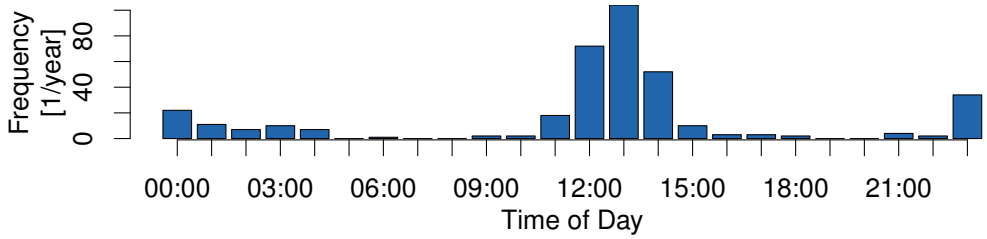

Figure 5.12 – Absolute frequencies for each hour of the day, having the minimal carbon intensity of that day.

Mean Cost by / Scheduling by	Day-Ahead Price [¢]		Carbon-Intensity [$gCO2$]	
Optimal Fixed Time	13:30	0.009 ± 0.008	13:00	0.467 ± 0.203
Default Fixed Time	00:00	0.013 ± 0.005	00:00	0.618 ± 0.211
Worst Fixed Time	19:30	0.021 ± 0.007	20:00	0.658 ± 0.197
Carbon-Aware		0.009 ± 0.007		0.420 ± 0.188
Price-Aware		0.006 ± 0.006		0.456 ± 0.205

Table 5.2 – Mean costs per workload execution with 1 standard deviation intervals. Values for fixed daily starting times of the Borg workload throughout the year 2023 compared to scheduling by CARBON-AWARE SYSTEMD TIMERS with grid carbon intensity and day-ahead prices as input metric.

For each type of scheduling, evaluated in this benchmark, I compare the mean cost per workload and mean carbon emissions per workload, as listed in Table 5.2. The costs are obtained from the simulated power consumption and the respective input metric, following Equations (5.1) and (5.2).

The carbon emissions for unaware scheduling range between $0.467 gCO2$ and $0.658 gCO2$ per workload run. CARBON-AWARE SYSTEMD TIMERS lower the mean emissions to $0.420 gCO2$ per workload, achieving a reduction of $10-36\%$. The present default configuration of the Borgmatic package in Debian schedules each backup run at 00:00 resulting in carbon emissions of $0.618 gCO2$ per workload execution. Compared to this default configuration, CARBON-AWARE SYSTEMD TIMERS reduce carbon emissions by 32%. If the interval, in which the workload is started each day, is not chosen optimal for German grid carbon intensity, the mean emissions for carbon-aware scheduling rise up to $0.425 gCO2$. This negligible difference illustrates, it is not necessary to choose the start interval region-dependent.

When operating CARBON-AWARE SYSTEMD TIMERS on day-ahead prices instead, the mean monetary cost is reduced by $36-72\%$, from $0.009-0.021$¢ to 0.006¢ per workload run. However, carbon emissions for the price-aware scheduling average at $0.456 gCO2$, ranking comparably to executing the workload at 13:00 every day. This shows, using day-ahead prices as input metric is not a suitable replacement for carbon-intensity forecasts. However, CARBON-AWARE SYSTEMD TIMERS are capable of using day-ahead prices to reduce monetary cost instead of carbon emissions. Similar to carbon emissions incurred by price-aware scheduling, the monetary cost per workload with carbon-aware scheduling yields no benefit over an optimal fixed starting time, with both methods resulting in a mean workload cost of 0.009¢.

5.4 Offline Timer Scheduling

All previous metric-aware tests rely on the availability of online forecast services. Should the use of online service be unavailable or undesirable for privacy concerns, CARBON-AWARE SYSTEMD TIMERS offer carbon-aware scheduling based on the STATIC CARBON PREDICTOR from Section 4.2. Scheduling by the fixed carbon-intensity table for Germany used by the STATIC CARBON PREDICTOR results in a cost of $0.467 gCO2$ per workload execution, with a standard deviation of $0.202 gCO2$. This reveals no benefit for the Borg test workload over a fixed-time scheduling at 13:00. However, the use of CARBON-AWARE SYSTEMD TIMERS, with either the STATIC CARBON PREDICTOR or a built-in fixed scheduling time for Germany, allows applications to schedule workloads at the optimal fixed time, without the need to configure each individual application according to properties of the local electricity grid, where the application is being used. The effort of providing region-specific configuration can be focused solely on CARBON-AWARE SYSTEMD TIMERS. Given the Borg test workload, this static scheduling still provides a reduction in carbon-emissions of up to 29% compared to a fixed scheduling at the worst starting time in Figure 5.11 with $0.658 gCO2$ per workload execution.

5.5 Evaluation Summary

CARBON-AWARE SYSTEMD TIMERS accomplish a 30% reduction in carbon-emissions in simulation of a daily test workload across the full year 2023 consisting of a Borg backup job with 45 minutes execution duration, when compared to the timer configuration currently present in the Borgmatic Debian package. When compared to a timer expiring at a fixed but optimal time of day, CARBON-AWARE SYSTEMD TIMERS still reduce carbon-emissions by 7%. Given the large, but regular, parts of daily variations in carbon intensity of the German electricity mix, simply scheduling daily timers at 13:00

instead of 00:00 also achieves a 24% reduction in carbon emissions for the Borg workload. The emission reductions, achievable by CARBON-AWARE SYSTEMD TIMERS, depend on the available electricity mix. While the German electricity mix allows for large emission reductions, the French energy mix, consisting mostly of nuclear power, offers little variation to be exploited, as shown by Wiesner et al. [Wie+21]. However, this region dependency is hidden from individual workloads, allowing utilization of carbon intensity variations, without region-dependent configuration of workloads.

6 RELATED WORK

Carbon-aware workload scheduling is a well covered topic in the context of datacenters. I cover four approaches to datacenter workload scheduling and how well each of them can be adapted to single-machine scenarios. Radovanović et al. present the CARBON-INTELLIGENT COMPUTING SYSTEM, a workload scheduling strategy deployed by Google's fleet of datacenters [Rad+22]. The approach combines power modeling, geographic shifting, and temporal shifting into Virtual Capacity Curves (VCC). Hönig et al. introduce ALBATROSS, an HPC cluster workload manager employing power modeling to achieve price awareness [Hön+18]. Aksanli et al. design a scheduler building VCCs from built-in green energy prediction [Aks+11]. Yang et al. present a queuing-network-based workload scheduler, combining geographic and temporal shifting to reduce carbon emissions [YHF22]. In summary, current approaches for carbon-aware workload scheduling focus on datacenters, with limited portability to single-machine scenarios. CARBON-AWARE SYSTEMD TIMERS bring the existing concept of temporal workload shifting to single-machine scenarios, while minimizing required configuration effort by software maintainers and machine administrators.

6.1 Carbon-Intelligent Computing System

Google's Carbon-Intelligent Computing System (CICS) combines power modeling, geographic shifting, and temporal shifting. Power modeling uses the heterogeneous hardware composition within each datacenter, as well as across different datacenters to allocate workloads to more powerful machines during low carbon intensity times and more efficient machines during high carbon intensity times. Geographic shifting exploits regional grid carbon intensity, by moving workloads to datacenters that, at the time, have access to low carbon intensity electricity. Finally, temporal shifting delays flexible workloads to a later time within the Limits of their SLA. The complexity of Google's fleet of datacenters is handled by combining these three metrics into VCCs. Each cluster is assigned a VCC which denotes a portion of the cluster's total computational capacity, the should be allocated at any point in time. Workloads are then scheduled across clusters according to their virtual capacities. CICS achieves $1-2\%$ reduction in total power consumption of the evaluated datacenters at times of high carbon intensity.

In a single-machine scenario, one could assign a VCC for the single host. The scheduling algorithm operating on the VCC would then degrade to temporal shifting. However, the overhead of setting up, as well as running a scheduler capable of handling a fleet of datacenters is unnecessary for a single machine. Furthermore, adhering to strict requirements of SLAs would not yield optimal results in a typical single-machine scenario, as shown in Section 3.3.

6.2 Albatross

The approach of ALBATROSS focuses on power modeling, moving workloads to more energy efficient nodes in heterogeneous clusters, if workload deadlines permit the incurred delays. ALBATROSS optimizes electricity price, not carbon intensity. Price optimization shares some similarities to carbon optimization. In particular, during times of high prices electricity consumption is reduced. During times of negative prices however, ALBATROSS has the option to intentionally increase electricity consumption, in order to make a profit. This behavior is not applicable to carbon awareness, as carbon intensity inherently can not be negative.

In a single-machine scenario, power modeling might be adapted into utility clamping, i.e. limiting CPU frequency to increase energy efficiency at times of high carbon intensity. However, Laurenzano et al. show the energy-optimal CPU frequency depends on the workload, with some workloads even suffering reduced energy efficiency at lower frequencies [Lau+11]. The energy savings observed by Laurenzano et al. average at 2.8%, compared to 7% and higher average carbon savings for CARBON-AWARE SYSTEMD TIMERS.

6.3 Green-Energy-Prediction Scheduler

The scheduler designed by Aksanli et al. predicts availability of green energy for a 30-minute interval and allocates computation time slots to consume any available green energy, similar to Virtual Capacity Curves. This requires a binary indicator for what is considered green energy. The approach manages to consume up to three times the portion of available green energy, compared to previous approaches.

In a scenario with no locally generated green energy, but only grid electricity associated with a carbon intensity value, this approach might be adapted to consume electricity, when carbon intensity drops below a threshold. To minimize carbon emissions, this approach would have to be further adapted for a larger prediction interval of at least several hour, as shown by [Wie+21].

6.4 Queuing-Network Scheduler

Yang et al. apply drift-plus-penalty optimization, in order to delay and distribute jobs to multiple datacenters. The approach minimizes emissions based on current carbon intensity values for each datacenter, while ensuring mean rate stability of their job queues. This scheduler achieves a 54% reduction in carbon emissions for scheduling AI-model-training tasks across 5 clouds.

If this approach is applied to a single-machine scenario, each job is delayed, based on the current carbon intensity and the number of jobs waiting to be executed. Since no carbon intensity predictions are taken into account, this scheduling algorithm degenerates to scheduling jobs when carbon intensity falls below a threshold or the number of queued jobs exceeds a maximum. The resulting greedy algorithm can not minimize carbon emissions.

6.5 Summary

Current approaches to carbon-aware workload scheduling focus solely on datacenters. While some approaches might be adaptable to a single-machine scenario, datacenter workload schedulers optimize under strict resource requirements for each workload. Adhering to these strict requirements yields non-optimal results in single-machine scenario, where workloads typically have neither minimal requirements for CPU availability, nor strict deadlines. Furthermore, scheduler implementations for datacenters would need to be heavily adapted for use with common single-machine system-management software, such as Systemd. CARBON-AWARE SYSTEMD TIMERS offer the benefits of carbon-aware temporal shifting to single-machine scenarios, exploiting the flexibility of typical workloads, and requiring minimal changes to workloads and the machine's system management.

7 CONCLUSION

Carbon Awareness is the concept of altering software behavior to reduce carbon emission incurred by running this software. While many approaches focus on bringing carbon-aware workload scheduling to datacenters, electricity consumption of consumer computers is at least comparable to datacenters. CARBON-AWARE SYSTEMD TIMERS introduce the benefits of carbon awareness to single-machine scenarios. Emission reductions for datacenter approaches vary between $1-2\%$ for total datacenter emissions and as much as 54% for isolated workload emissions in specific benchmarks. CARBON-AWARE SYSTEMD TIMERS reduce emissions of the Borg backup workload by at least 10% and up to 36% depending on the previous configuration of the workload.

The potential of CARBON-AWARE SYSTEMD TIMERS can be further increased by better availability of carbon-intensity forecasts, introducing system load prediction, and expanding the presented interface to control long-running interruptible services and manually set up workloads.

Some countries, such as Great Britain already offer publicly funded, freely-available carbon-intensity forecasts [Nat17]. Widespread free availability of such forecasts would allow carbon awareness for small scale use cases, that can not economically make use of non-free services, such as Electricity Maps.

Adding system load prediction to CARBON-AWARE SYSTEMD TIMERS could reduce interference with interactive use, as well as lower the impact of resource contention between carbon-aware timers. This can be achieved by collecting more detailed resource-usage data on carbon-aware, as well as non-aware services, combined with prediction algorithm to forecast resource usage.

The infrastructure built by CARBON-AWARE SYSTEMD TIMERS can be expanded to manage long-running interruptible services, such as Folding@Home [Fol00]. This would allow contributing private computational capacity to projects that serve the public interest, while remaining conscious of the resulting carbon emissions.

Systemd has the capability to manually run workloads via `systemd-run`, without setting up a Systemd service first. Integrating `systemd-run` with CARBON-AWARE SYSTEMD TIMERS would allow manually-started workloads to be made carbon-aware with no further configuration.

In conclusion, CARBON-AWARE SYSTEMD TIMERS can significantly reduce emissions for suitable workloads, while requiring minimal setup effort from software maintainers or system administrators. With suitable Systemd timers marked by workload package maintainers, installing the set of CARBON-AWARE SYSTEMD TIMER Debian packages, suffices to make these timers carbon-aware. No region-dependent configuration of individual workloads is necessary.

LIST OF ACRONYMS

CICS Carbon-Intelligent Computing System

CO_2 carbon-dioxide

E electricity consumption

I carbon-intensity

O operational carbon

RAPL Running Average Power Limit

SLA Service Level Agreements

VCC Virtual Capacity Curves

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

REFERENCES

[Aks+11] Baris Aksanli et al. “Utilizing green energy prediction to schedule mixed batch and service jobs in data centers.” In: *Proceedings of the 4th workshop on power-aware computing and systems*. 2011, pp. 1–5.

[Bel05] Fabrice Bellard. “QEMU, a fast and portable dynamic translator.” In: *USENIX annual technical conference, FREENIX Track*. Vol. 41. 46. California, USA. 2005, pp. 10–5555.

[Bor15] BorgBackup Project. *Deduplicating archiver with compression and encryption*. Retrieved: 2024-11-02. 2015. URL: https://web.archive.org/web/20241007122727/https://www.borgbackup.org/.

[CA13] Peter Corcoran and Anders Andrae. “Emerging trends in electricity consumption for consumer ICT.” In: *National University of Ireland, Galway, Connacht, Ireland, Tech. Rep* (2013).

[Ele22] Electricity Maps. *Real-time and predictive electricity signals*. Retrieved: 2024-11-02. 2022. URL: https://web.archive.org/web/20241002163656/https://electricitymaps.com.

[ENT24a] ENTSO-E. *Central collection and publication of electricity generation, transportation and consumption data and information for the pan-European market*. Retrieved: 2024-11-02. 2024. URL: https://web.archive.org/web/20240826110855/https://transparency.entsoe.eu/dashboard/show.

[ENT24b] ENTSO-E. *Transparency Platform RESTful API*. Retrieved: 2024-11-02. 2024. URL: https://web.archive.org/web/20240912223927/https://transparency.entsoe.eu/content/static_content/Static%20content/web%20api/Guide.html.

[Fol00] Folding@Home. *Distributed Computing Project for Simulating Protein Dynamics*. Retrieved: 2024-11-02. 2000. URL: https://web.archive.org/web/20241102230301/https://foldingathome.org/about-2/.

[Gre24] Green Software Foundation. *Software Carbon Intensity (SCI) Specification*. Retrieved: 2024-11-02. 2024. URL: https://web.archive.org/web/20241005110622/https://sci.greensoftware.foundation/.

[Hön+18] Timo Hönig et al. “How to make profit: Exploiting fluctuating electricity prices with Albatross, a runtime system for heterogeneous HPC clusters.” In: *Proceedings of the 8th International Workshop on Runtime and Operating Systems for Supercomputers*. 2018, pp. 1–9.

[Ibr+16] Shadi Ibrahim et al. “Governing energy consumption in Hadoop through CPU frequency scaling: An analysis.” In: *Future Generation Computer Systems* 54 (2016), pp. 219–232.

[Kha+18] Kashif Nizam Khan et al. “Rapl in action: Experiences in using rapl for power measurements.” In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 3.2 (2018), pp. 1–26.

[Kiv+07] Avi Kivity et al. “kvm: the Linux virtual machine monitor.” In: *Proceedings of the Linux symposium*. Vol. 1. 8. Dttawa, Dntorio, Canada. 2007, pp. 225–230.

[Lau+11] Michael A Laurenzano et al. “Reducing energy usage with memory and computation-aware dynamic frequency scaling.” In: *Euro-Par 2011 Parallel Processing: 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29-September 2, 2011, Proceedings, Part I 17*. Springer. 2011, pp. 79–90.

[Lin15] Linux Kernel. *Control Group v2*. Retrieved: 2024-11-02. 2015. URL: https://web.archive.org/web/20241007035537/https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html.

[Lin24] Linux Kernel. *perf: Linux profiling with performance counters*. Retrieved: 2024-11-02. 2024. URL: https://web.archive.org/web/20241006221454/https://perf.wiki.kernel.org/index.php/Main_Page.

[Mas+20] Eric Masanet et al. “Recalibrating global data center energy-use estimates.” In: *Science* 367.6481 (2020), pp. 984–986.

[Nat17] National Energy System Operator (NESO). *Carbon Intensity API for Great Britain*. Retrieved: 2024-11-02. 2017. URL: https://web.archive.org/web/20241007125324/https://www.carbonintensity.org.uk/.

[Rad+22] Ana Radovanović et al. “Carbon-aware computing for datacenters.” In: *IEEE Transactions on Power Systems* 38.2 (2022), pp. 1270–1280.

[sys24] systemd. *System and Service Manager*. Retrieved: 2024-11-02. 2024. URL: https://web.archive.org/web/20241009202937/https://systemd.io/.

[Wie+21] Philipp Wiesner et al. “Let’s Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud.” In: *22nd International Middleware Conference* (Dec. 2021). DOI: 10.1145/3464298.3493399.

[YHF22] Chien-Sheng Yang, Chien-Chun Huang-Fu, and I-Kang Fu. “Carbon-neutralized task scheduling for green computing networks.” In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE. 2022, pp. 4824–4829.

APPENDIX A

A.1 Debian packages containing systemd timers

Package	Task	Duration	Frequency	Timer Type
acmetool	updates	Short	Day	Wall
aide-common	notifications+data checks	Short	Day	Wall
amazon-ec2-net-utils	updates	Short	Minute	Monotonic
anacron	task scheduling	Short	Hour	Wall
apt	updates	Long	Day	Wall
apticron-systemd	notifications	Short	Day	Wall
apt-listbugs	updates	Short	Hour	Wall+Monotonic
apt-show-versions	updates	Short	Day	Wall
atop	service restart	Short	Day	Wall
borgmatic	backup	Long	Day	Wall
btrbk	backup	Long	Day	Wall
btrfsmaintenance	data checks	Long	Month	Wall
burp	backup	Long	Hour	Wall
certbot	updates	Short	Day	Wall
chkrootkit	notifications+data checks	Short	Day	Monotonic
chrony	updates	Short	Hour	Monotonic
cloudflare-ddns	updates	Short	Minute	Monotonic
dde-calendar	notifications	Short	Minute	Monotonic
ddupdate	updates	Short	Hour	Monotonic
debci-collector	updates	Short	Minute	Wall
debian-edu-config	data checks+updates	Short	Minute-Day	Monotonic
debspawn	data checks	Short	Month	Wall
diffmon	notifications+data checks	Short	Day	Wall
dpkg	backup	Short	Day	Wall
drkonqi	data checks	Short	Week	Wall
duply	backup	Long	Day	Monotonic
e2fsprogs	data checks	Long	Week	Wall
etckeeper	data checks	Short	Day	Monotonic
exim4-base	notifications+data checks	Short	Day	Monotonic
fetch-crl	updates	Short	Hour	Monotonic
foomuuri	updates	Short	Minute	Monotonic
freedombox	task scheduling	Short	Minute	Wall
freeipa-client-epn	notifications	Short	Day	Wall
freeipa-healthcheck	notifications	Short	Day	Wall

fwupd	updates	Short	Day	Wall
geoipupdate	updates	Short	Week	Wall
google-compute-engine-oslogin	data checks	Short	Hour	Monotonic
josm	updates	Short	Day	Wall
kanboard	notifications	Short	Day	Wall
laptop-mode-tools	notifications	Short	Minute	Monotonic
liblemonldap-ng-handler-perl	data checks	Short	Hour	Wall
liblemonldap-ng-portal-perl	data checks	Short	Hour	Wall
location	updates	Short	Day Wall	
logrotate	data checks	Short	Day	Wall
lynis	data checks	Short	Day	Wall
man-db	data checks	Short	Day	Wall
mdadm	data checks	Short	Day	Wall
ntpsec	data checks	Short	Day	Wall
ntpsec-ntpviz	statistics	Short	Minute	Monotonic
ntpsec-ntpviz	data checks	Short	Day	Wall
ocsinventory-agent	data checks+updates	Long Day	Wall	
offlineimap3	updates	Short	Minute	Monotonic
openqa	data checks	Short	Hour-Week	Wall
pagure	notifications	Short	Day	Wall
pagure	updates	Short	Hour	Wall
painintheapt	notifications	Short	Day-Week	Wall
pcp	statistics	Short	Minute	Monotonic
pcp	data checks	Short	Hour-Day	Wall
pgcluu	statistics	Short	Minute	Monotonic
php-common	data checks	Short	Hour	Wall
pk4	data checks	Short	Minute	Monotonic
plocate	data checks	Short	Day	Wall
pnopaste	data checks	Short	Day	Wall
podman	updates	Long	Day	Wall
postgresql-common	backup+data checks	Long	Day-Week	Wall
privoxy	data checks	Short	Day	Wall
profile-sync-daemon	data checks	Short	Hour	Monotonic
prometheus-node-exporter-collectors	data checks	Short	Minute	Monotonic
rancid	data checks	Short	Hour-Day	Wall
roundcube-core	data checks	Short	Hour-Day	Wall
rpki-client	data checks	Short	Hour	Wall
sanoid	data checks	Short	Minute	Wall
sitesummary	data checks	Short	Day	Wall
sitesummary-client	data checks	Short	Day	Wall
snapper	data checks	Short	Minute-Day	Monotonic+Wall
spamassassin	data checks	Short	Day	Wall
sysstat	statistics+notifications	Short	Minute-Day	Wall
systemd	data checks	Short	Day	Monotonic
systemd-cron	task scheduling	Short	Hour-Year	Wall
tuptime	data checks	Short	Minute	Wall
util-linux	data checks	Long	Week	Wall
vdirsyncer	updates	Short	Minute	Monotonic
ypserv	updates	Short	Hour-Day	Wall
zfsutils-linux	data checks	Long	Week-Month	Wall

Table A.1 – Debian packages with systemd timers